Unity 팀 프로젝트 개발문서

Monster & Behaviour Tree

작성자 : 한창신

1. 기획내용

1-1. 기획팀 기획 내용

- 몬스터 기획서

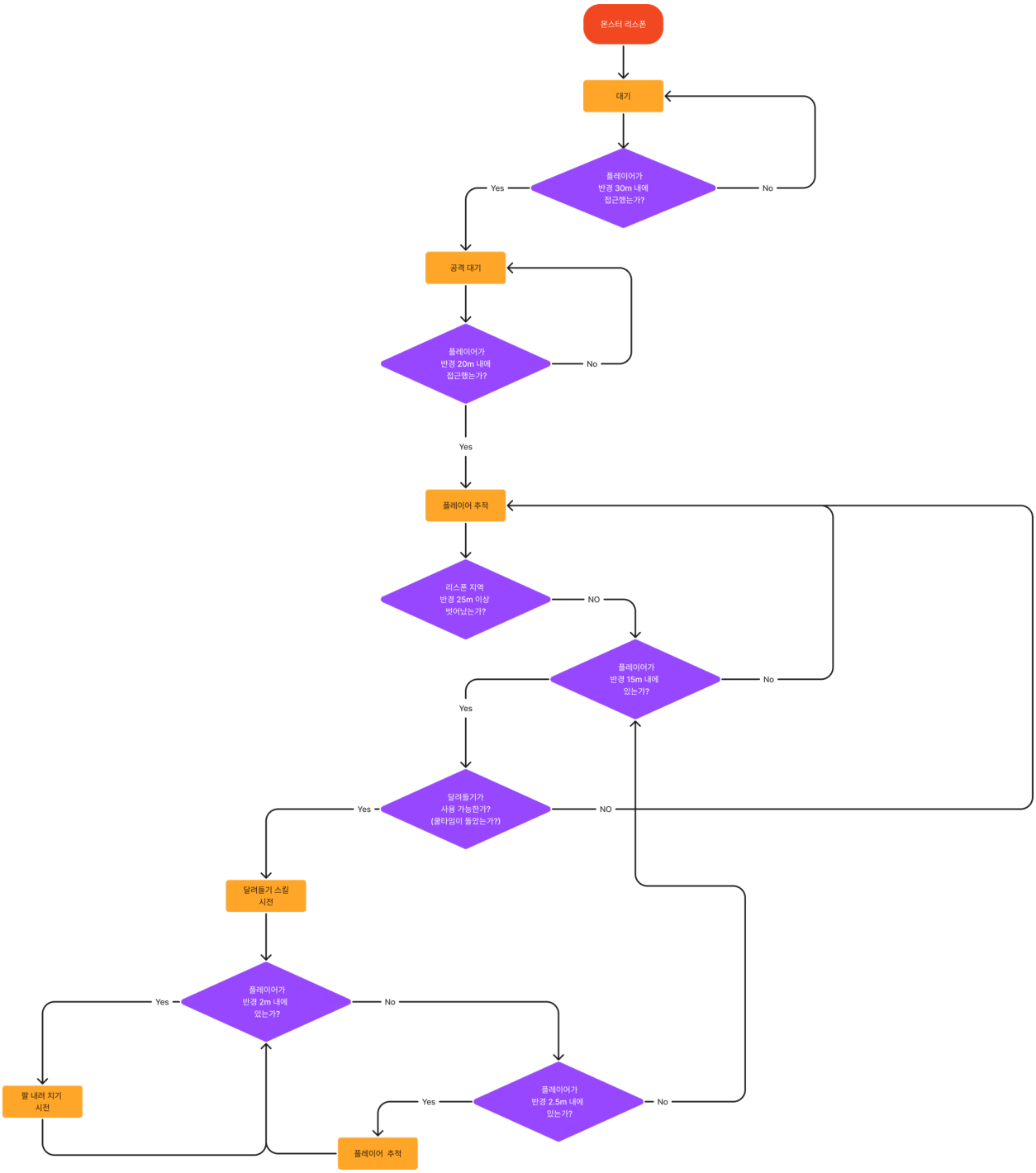
평범한 스켈레톤		
대분류	소분류	설명
정보	체력	5
	공격력	1
	공격 범위	中
	공격속도	2
	기본 이동속도	2m/s
속성	등급	일반
	타입	지상
	계열	해골 계열
	속성	무속성
	공격 방식	근거리
	공격 타입	선공
	등장장소	미정 (추후 맵 좌표 값으로 설정할 것)
	서식지	던전 내 아무 곳
	기획특성	던전 곳곳에서 흔히 등장하는 몬스터
	습성	평범하고 느리게 움직이는 해골 이미지
	배경 스토리	마법사의 무덤 근처에서 죽은 자들의 영혼은 마석의 마력에 영혼이 묶여 시체에 남게 된다. 그들은 처음엔 불멸을 얻은 줄 알고 좋아했으나 육체는 점점 썩어 없어지고 잠들지도 먹지도 못한 채 시간을 보냈다. 시간이 지나 그들은 이성을 잃고 미쳐버리며 마법사의 무덤 근처를 돌아다니는 몬스터로 남게 되었다.
	행동	주변에 인기척이 들리면 움직이기 시작한다. 관절 하나, 하나 삐걱거리며 움직인다. 움직임이 멈추면 갑자기 달려든다.
패턴		설명
공격	1:팔 내려치기	팔로 적을 내려쳐 플레이어에게 1의 피해를 입히고 악경직 상태로 만든다.
이동	1: 달려들기	플레이어를 향해 4m/s로 달려간다.
애니메이션		설명
기본대기		누워있는 상태
공격대기	1	누워있다가 플레이어가 근접 및 공격 시 일어나서 정지 상태
	2	근접 시 서서히 맵을 돌아다닌다
이동	1	천천히 플레이어를 향해 걸어 온다.
	2	갑자기 팔을 들어 올리며 플레이어를 향해 달려든다.
공격	1	두 팔을 들어 올렸다 몸과 함께 내려치며 공격한다.
맞는 동작	1	고개가 뒤로 젖혀지며 팔을 들어 올라가고 몸 전체가 뒤로 약간 밀리는 형태
사망	1	뼈들이 우수수 떨어지는 부서지는 형태

모험가 스켈레톤		
대분류	소분류	설명
정보	체력	5
	공격력	1
	공격범위	中
	공격속도	2
	기본 이동속도	2m/s
속성	등급	일반
	타입	지상
	계열	해골 계열
	속성	무속성
	공격 방식	근거리
	공격 타입	선공
	등장장소	미정 (추후 맵 좌표 값으로 설정할 것)
	서식지	탐사구역 (필드 외곽)
	기획특성	탐사구역에서 등장하는 근거리 해골계열 몬스터
	습성	평범하고 느리게 움직이는 해골에 장비가 입혀진 이미지 평범한 스켈레톤 보다 단단하고 위협적인 공격을 한다.
	배경 스토리	마법사의 무덤 근처에서 죽은 모험가들의 영혼은 마석의 마력에 영혼이 묶여 시체에 남게 된다. 죽은 줄도 모르고 던전을 탐험하던 그들의 육체는 점점 썩어 없어지고 잠들지도 먹지도 못한 채 시간을 보냈다. 시간이 지나 그들은 이성을 잃고 미쳐버리며 마법사의 무덤 근처를 돌아다니는 몬스터로 남게 되었다.
	행동	주변에 인기척이 들리면 움직이기 시작한다. 관절 하나, 하나 삐걱거리며 움직인다. 움직임이 멈추면 갑자기 달려든다.
패턴		설명
공격	1:팔 내려치기	팔로 적을 내려쳐 플레이어에게 1의 피해를 입히고 악경직 상태로 만든다.
	2: 흉기 휘두르기	오른 손에 들고있는 무기를 가로로 휘두르며 플레이어에게 1의 피해를 입히고 악경직 상태로 만든다.
이동	1: 달려들기	플레이어를 향해 4m/s로 달려간다.
애니메이션		설명
기본대기		누워있는 상태
공격대기	1	누워있다가 플레이어가 근접 및 공격 시 일어나서 정지 상태
	2	근접 시 서서히 맵을 돌아다닌다
이동	1	천천히 플레이어를 향해 걸어 온다. (기본 이동)
	2	갑자기 팔을 들어 올리며 플레이어를 향해 달려든다.
공격	1	두 팔을 들어 올렸다 내려쳐 공격한다.
	2	오른 손에 들고 있는 무기를 앞을 향해 휘두르며 공격한다.
맞는 동작	1	고개가 뒤로 젖혀지며 팔을 들어 올라가고 몸 전체가 뒤로 약간 밀리는 형태
사망	1	뼈들이 우수수 떨어지는 부서지는 형태

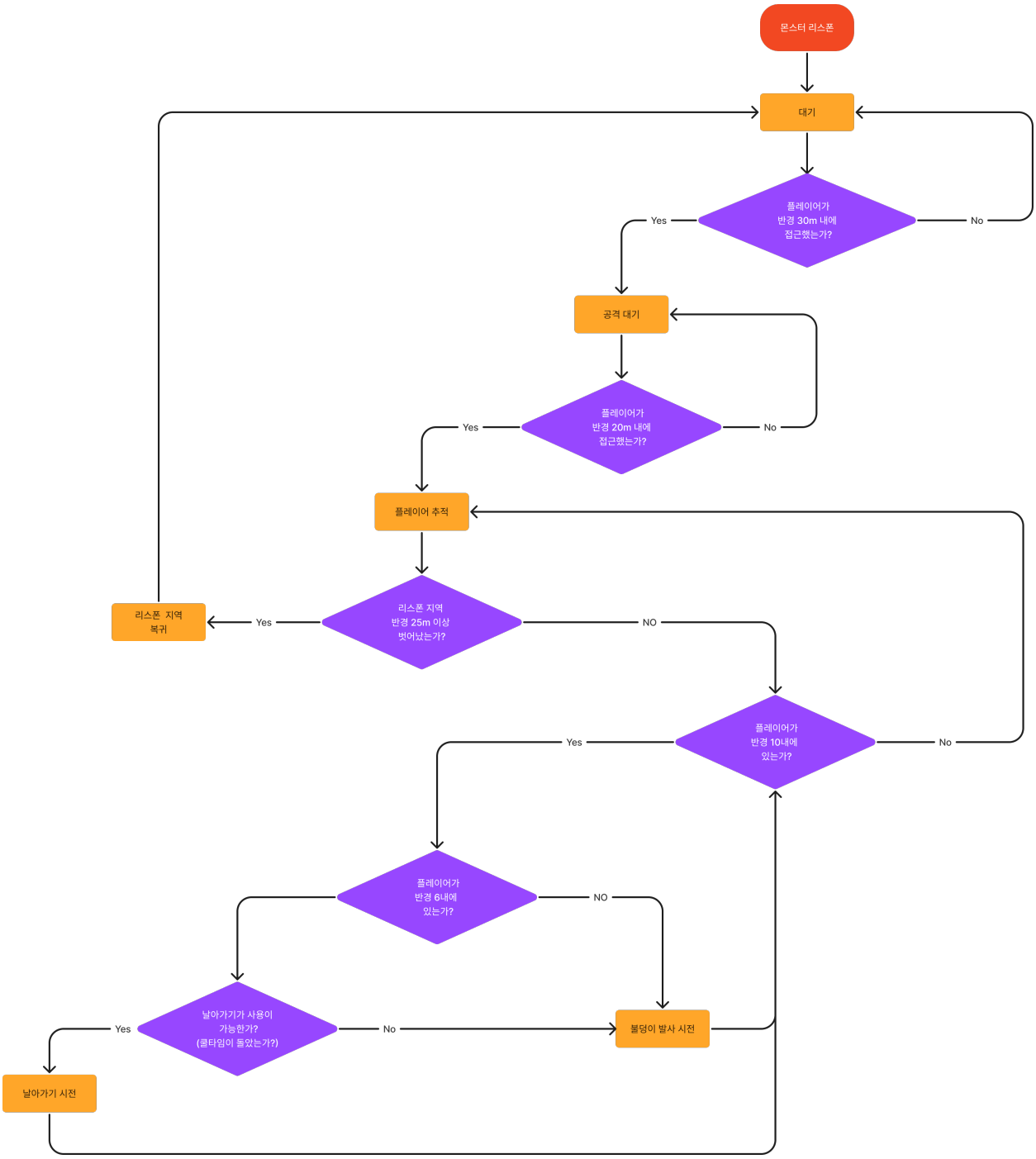
마법사 스켈레톤		
대분류	소분류	설명
정보	체력	3
	공격력	1
	공격범위	中
	공격속도	3
	기본 이동속도	3m/s
속성	등급	일반
	타입	공중
	계열	해골 계열
	속성	무속성
	공격 방식	마법
	공격 타입	선공
	등장장소	미정 (추후 맵 좌표 값으로 설정할 것)
	서식지	탐사구역 (필드 외곽)
	기획특성	탐사구역에서 등장하는 마법공격 해골계열 몬스터
	습성	평범하고 느리게 움직이는 해골에 마법사 장비가 입혀진 이미지 다른 스켈레톤과 달리 마법 공격을 사용한다. 마법으로 몸을 띄워 다른 해골들에 비해 움직임이 빠르다.
	배경 스토리	오래전 흑마법사의 성에서 '흑마법사 제레프'와 전투를 하다 목숨을 잃은 마법사들이 마석의 마력에 영혼이 묶여 시체에 남게 된다. 그들은 죽어서도 제레프와 맞서 싸웠지만 끝내 패배하고 다음을 기약하며 언데드로 살게된다. 하지만 그들의 육체는 점점 썩어 없어지고 잠들지도 먹지도 못한 채 시간을 보냈다. 시간이 지나 그들은 이성을 잃고 미쳐버리며 마법사의 무덤 근처를 돌아다니는 몬스터로 남게 되었다.
	행동	주변에 인기척이 들리면 움직이기 시작한다. 몸이 붕 뜨며 하늘을 떠다닌다. 움직임이 멈추면 갑자기 마법을 사용한다.
패턴		설명
공격	1:불덩이 발사	2초 간 마력을 모아 플레이어를 향해 1에 피해를 주는 불덩이를 10m 만큼 발사한다.
이동	1: 날아가기	플레이어가 일정 범위 다가올 경우 4m 뒤로 날아간다.
애니메이션		설명
기본대기		누워있는 상태
공격대기	1	누워있다가 플레이어가 근접 및 공격 시 공중에서 뜬 상태에서 정지
	2	근접 시 서서히 맵을 돌아다닌다
이동	1	공중에 떠다니며 플레이어 근처에서 거리를 둔다.
	2	플레이어가 일정 범위 다가올 경우 거리를 벌리며 날아간다.
마법시전	1	두 팔을 내밀며 불덩이를 발사한다.
맞는 동작	1	고개가 뒤로 젖혀지며 팔을 들어 올라가고 몸 전체가 뒤로 약간 밀리는 형태
사망	1	공중에서 뼈들이 우수수 떨어지는 부서지는 형태

공대원 스켈레톤		
대분류	소분류	설명
정보	체력	7
	공격력	2
	공격범위	中
	공격속도	3
	기본 이동속도	3m/s
관련 돌멩이	약점 돌멩이	죽발 돌멩이, 불 속성 돌멩이, 날카로운 돌멩이
	무시 돌멩이	
속성	등급	일반
	타입	지상
	계열	해골 계열
	속성	무속성
	공격 방식	근거리
	공격 타입	선공
	등장장소	미정 (추후 맵 좌표 값으로 설정할 것)
	서식지	탐사구역 (필드 외곽)
	기획특성	탐사구역에서 등장하는 근거리 해골계열 몬스터
	습성	모험가 스켈레톤보다 강력한 모험가 스켈레톤 모험가 스켈레톤 보다 단단하고 강한 공격을 한다.
	배경 스토리	마법사의 무덤 깊은 곳에서 죽은 모험가들의 영혼은 마석의 마력에 영혼이 묶여 시체에 남게 된다. 죽은 줄도 모르고 던전을 탐험하던 그들의 육체는 점점 썩어 없어지고 잠들지도 먹지도 못한 채 시간을 보냈다. 시간이 지나 그들은 이성을 잃고 미쳐버리며 마법사의 무덤 근처를 돌아다니는 몬스터로 남게 되었다.
	행동	주변에 인기척이 들리면 움직이기 시작한다. 관절 하나, 하나 삐걱거리며 움직인다. 움직임이 멈추면 갑자기 달려든다.
패턴		설명
공격	1:팔 내려치기	팔로 적을 내려쳐 플레이어에게 3의 피해를 입히고 악경직 상태로 만든다.
이동	1: 달려들기	플레이어를 향해 5m/s로 달려간다.
애니메이션		설명
기본대기		누워있는 상태
공격대기	1	누워있다가 플레이어가 근접 및 공격 시 일어나서 정지 상태
	2	근접 시 서서히 맵을 돌아다닌다
이동	1	천천히 플레이어를 향해 걸어 온다. (기본 이동)
	2	갑자기 팔을 들어 올리며 플레이어를 향해 달려든다.
공격	1	두 팔을 들어 올렸다 내려쳐 공격한다.
맞는 동작	1	고개가 뒤로 젖혀지며 팔을 들어 올라가고 몸 전체가 뒤로 약간 밀리는 형태
사망	1	뼈들이 우수수 떨어지는 부서지는 형태

- 근접공격 다이어그램


몬스터 리스폰
대기
플레이어가
반경 30m 내에
접근했는가?
Yes
No
공격 대기
플레이어가
반경 20m 내에
접근했는가?
No
Yes
플레이어 추적
리스폰 지역
반경 25m 이상
벗어났는가?
NO
플레이어가
반경 15m 내에
있는가?
No
Yes
달려들기가
사용 가능한가?
(쿨타임이 돌았는가?)
Yes
NO
달려들기 스킬
시전
플레이어가
반경 2m 내에
있는가?
Yes
No
팔 내려 치기
시전
플레이어가
반경 2.5m 내에
있는가?
Yes
No
플레이어 추적

- 원거리 몬스터 다이어그램



1-2. 개발팀 기획 요청 내용

1-2.1 몬스터 공통 Stat

몬스터 공통 스텟 내용						
구분	값				자료형	기타
	평범한 스켈레톤	모험가 스켈레톤	마법사 스켈레톤	공대원 스켈레톤		
체력					float	최대(시작) 체력
속도					float	일반 걷기 속도 (초당 미터)
회전 속도					float	몬스터 초당 n도 회전
플레이어 감지					float	미터 단위
플레이어 추격 감지					float	
초과이동 거리					float	
스폰 지역 복귀 속도					float	초당 미터
플레이어 추적 멈춤 거리					float	몬스터가 추적 완료되는 거리
약점 비율					float	약점 맞을 시 데미지의 몇배 받는지
리젠 시간					float	죽은 후 리젠 타임
드랍 아이템					list<int>	아이템 드랍 번호 입력

1-2.2 몬스터 공격

- 공격 여부 표시

스킬		몬스터			
기획팀	개발팀	평범한 스켈레톤	모험가 스켈레톤	마법사 스켈레톤	공대원 스켈레톤
달려들기	RunToPlayer	O	O		O
팔내려치기	MeleeAttack	O	O		O
파이어볼	Fireball			O	
달아나기	Flee			O	
무기 공격	WeaponAttack		O		O

- 근접공격

몬스터 근접 공격 내용						
구분	값				자료형	기타
	평범한 스켈레톤	모험가 스켈레톤	마법사 스켈레톤	공대원 스켈레톤		
데미지					int	
쿨타임					float	
공격 가능 거리					float	몬스터가 플레이어와의 거리가 얼만큼 되어야 공격을 하는지
공격 지속 시간					float	공격이 얼만큼 지속되는지
애니메이션 대기 시간					float	공격 애니메이션을 실행 후 공격하는 타이밍
공격 위치					Vector3	몬스터 기준 공격이 되는 위치(몬스터 기준 x, y, z 미터 단위)
공격 범위					Vector3	공격위치 범위 x, y, z (미터단위)
* x : 몬스터 오른쪽, y : 몬스터 위쪽, z : 몬스터전방						
* 공격 위치와 범위 유니티에서 보이게 해드리니 그거 보시고 작성하시면 편하실거에요!						

- 달려들기

몬스터 달려들기						
구분	값				자료형	기타
	평범한 스켈레톤	모험가 스켈레톤	마법사 스켈레톤	공대원 스켈레톤		
쿨타임					int	
공격 가능 거리					float	
공격 지속 시간					float	
속도					float	달려들기 속도
멈춤 거리					float	
속도는 기존 속도의 배율로 해도 되고 속도 값을 넣어주셔도 됩니다. 표시만 부탁드려요						

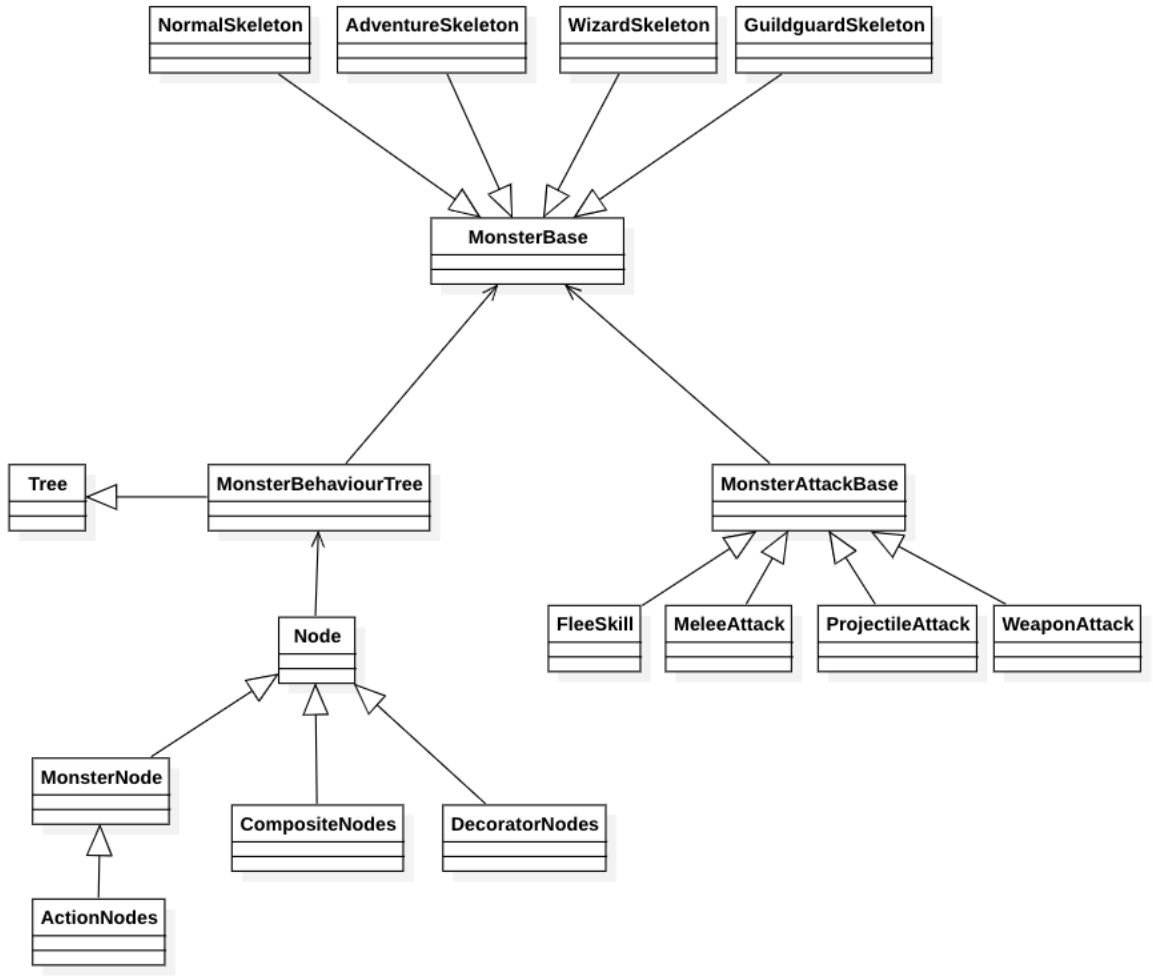
1-2.3 몬스터 행동 우선순위

	구분	평범한 스켈레톤	모험가 스켈레톤	마법사 스켈레톤	공대원 스켈레톤
우선순위	플레이어 추적	4	5	4	5
	초과이동 복귀	1	1	1	1
	달려들기	3	4		4
	팔내려치기	2	3		3
	파이어볼			2	
	달아나기			3	
	무기공격		2		2

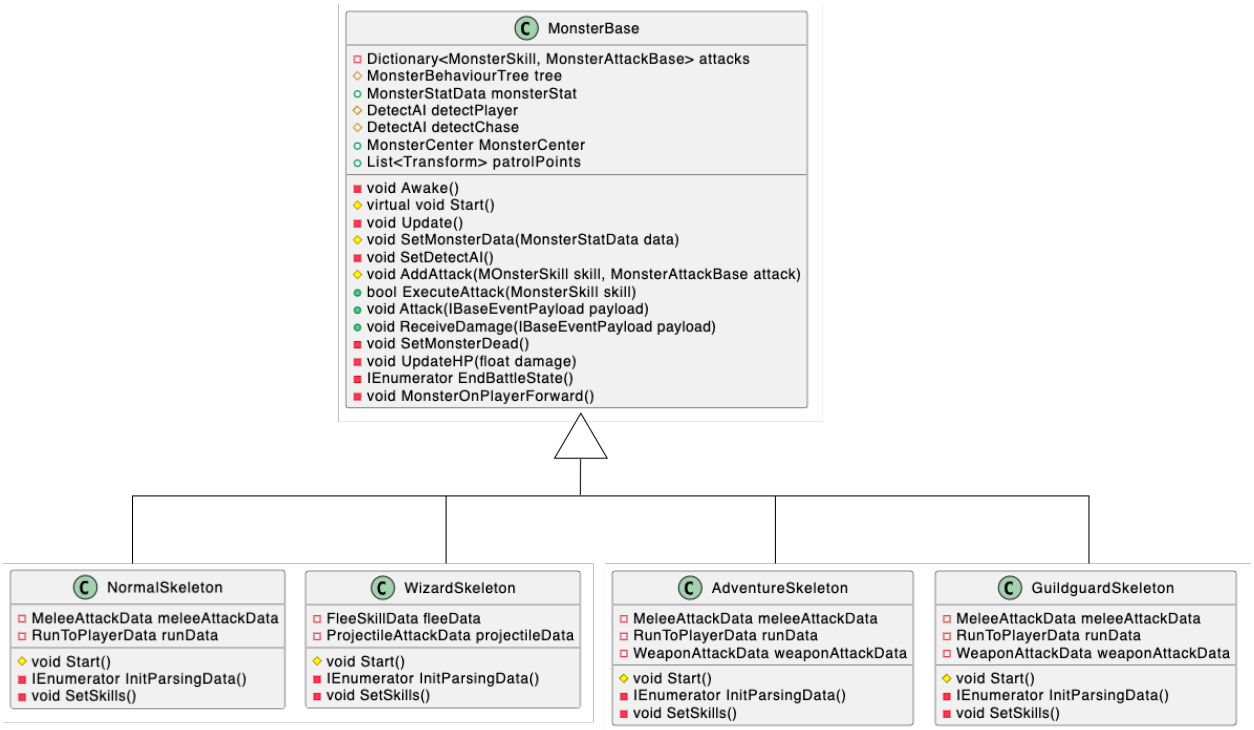
* 몬스터가 하는 행동에 우선순위가 필요합니다. 예를 들어 몬스터가 플레이어에게 달려들기와 팔 내려치기를 동시에 할 수 있을 때 어느 행동을 먼저 할지 정해주시면 그에 맞게 개발 할 수 있습니다.
* 제가 개발하면서 우선순위 임의로 정했고, 확인해보시고 수정내용 있으시면 말씀 부탁드립니다.

2. 몬스터 설계

2-1. Diagram



2-2. Monster



2-2.1 MonsterBase

```
using System.Collections;
using System.Collections.Generic;
using Assets.Scripts.Combat;
using Assets.Scripts.Managers;
using Assets.Scripts.Monsters.Utility;
using Assets.Scripts.UI.Monster;
using Assets.Scripts.Utils;
using Channels.Combat;
using Channels.Components;
using Channels.Type;
using UnityEngine;
using UnityEngine.AI;

namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterBase : MonoBehaviour, ICombatant
    {
        private Dictionary<MonsterSkill, MonsterAttackBase> attacks;
        [SerializeField] protected MonsterBehaviourTree tree;
        public MonsterStatData monsterStat;
        protected DetectAI detectPlayer;
        protected DetectAI detectChase;
        [SerializeField] public MonsterCenter MonsterCenter { get; private set; }
        [SerializeField] public List<Transform> patrolPoints;

        private NavMeshAgent agent;
        private MonsterAudioController audioController;
        private Animator animator;

        private TicketMachine ticketMachine;

        private float currentHP;
        private bool isHeadshot;

        private UIMonsterBillboard billboard;
```

```
        private Transform billboardObject;
        private bool isBillboardOn;
        private Transform cameraObj;
        private Coroutine battleCoroutine;
        private readonly MonsterDataContainer dataContainer = new();

        private void Awake()
        {
            audioController = GetComponent<MonsterAudioController>();
            animator = GetComponent<Animator>();

            ticketMachine = gameObject.GetOrAddComponent<TicketMachine>();
            ticketMachine.AddTickets(ChannelType.Combat, ChannelType.Monster);

            SetDetectAI();

            tree = GetComponent<MonsterBehaviourTree>();
        }
        protected virtual void Start()
        {
            tree.AddMonsterData<GameObject>(MonsterData.v3SpawnPosition, MonsterCenter.Player);
        }
        private void Update()
        {
            MonsterOnPlayerForward();
        }

        // >> : Set Datas
        protected void SetMonsterData(MonsterStatData data)
        {
            // UI
            currentHP = monsterStat.HP;
            dataContainer.MaxHp = (int)monsterStat.HP;
            dataContainer.PrevHp = (int)monsterStat.HP;
            dataContainer.CurrentHp.Value = (int)currentHP;
            dataContainer.Name = monsterStat.name;

            billboardObject = Functions.FindChildByName(gameObject, "Billboard").transform;
            cameraObj = Camera.main.transform;
            billboard = UIManager.Instance.MakeStatic<UIMonsterBillboard>(billboardObject,
UIManager.UIMonsterBillboard);
            HideBillboard();
            billboard.InitBillboard(billboardObject);

            // Agent
            agent.speed = monsterStat.speed;

            billboard.InitData(dataContainer);
        }
        private void SetDetectAI()
        {
            GameObject detectPlayerObj = new GameObject("DetectPlayer");
            detectPlayerObj.transform.SetParent(transform);
            detectPlayerObj.transform.localPosition = Vector3.zero;
            detectPlayerObj.transform.localRotation = Quaternion.Euler(Vector3.zero);
            detectPlayerObj.transform.localScale = Vector3.one;
            detectPlayer = detectPlayerObj.AddComponent<DetectAI>();

            GameObject detectChaseObj = new GameObject("DetectChase");
            detectChaseObj.transform.SetParent(transform);
            detectChaseObj.transform.localPosition = Vector3.zero;
            detectChaseObj.transform.localRotation = Quaternion.Euler(Vector3.zero);
            detectChaseObj.transform.localScale = Vector3.one;
            detectPlayer = detectChaseObj.AddComponent<DetectAI>();
            detectChaseObj.AddComponent<SphereCollider>();
        }
        protected void AddAttack(MonsterSkill skill, MonsterAttackBase attack)
        {
            GameObject attackObj = new GameObject(skill.ToString());
```

```
            attackObj.transform.SetParent(transform);
            attackObj.transform.localPosition = Vector3.zero;
            attackObj.transform.localRotation = Quaternion.Euler(Vector3.zero);
            attackObj.transform.localScale = Vector3.one;

            switch (skill)
            {
                case MonsterSkill.Melee:
                    attack = attackObj.AddComponent<MonsterMeleeAttack>();
                    break;
                case MonsterSkill.Projectile:
                    attack = attackObj.AddComponent<MonsterProjectileAttack>();
                    break;
                case MonsterSkill.Weapon:
                    attack = attackObj.AddComponent<MonsterWeaponeAttack>();
                    break;
                case MonsterSkill.Flee:
                    attack = attackObj.AddComponent<MonsterFleeSkill>();
                    break;
            }
            if (attack == null)
            {
                Debug.LogFormat("{0} Failed To Add Skill {1}", transform.name, skill.ToString());
                return;
            }

            attacks.Add(skill, attack);
        }

        // >> : Battles
        public bool ExecuteAttack(MonsterSkill skill)
        {
            MonsterAttackBase attack;
            if (attacks.TryGetValue(skill, out attack))
            {
                attack.ExecuteAttack();
                return true;
            }

            Debug.LogFormat("Trying To Access Attack Does Not Have : {0}, {1}", transform.name,
skill.ToString());
            return false;
        }
        public void Attack(IBaseEventPayload payload)
        {
            CombatPayload combatPayload = payload as CombatPayload;
            ticketMachine.SendMessage(ChannelType.Combat, combatPayload);
        }

        public void ReceiveDamage(IBaseEventPayload payload)
        {
            CombatPayload combatPayload = payload as CombatPayload;
            UpdateHP(combatPayload.Damage);
        }

        // >> : MonsterDamaged or Dead
        private void SetMonsterDead()
        {
            audioController.PlayAudio(MonsterAudioType.Dead);
            if (animator == null)
            {
                animator = GetComponent<Animator>();
            }
            animator.Play("Dead");
            GetComponent<Collider>().enabled = false;
            MonsterPayload monsterPayload = new();
            monsterPayload.RespawnTime = monsterStat.respawnTime;
            monsterPayload.Monster = transform;
            monsterPayload.ItemDrop = monsterStat.itemTableNum;
```

```
            ticketMachine.SendMessage(ChannelType.Monster, monsterPayload);
        }
        private void UpdateHP(float damage)
        {
            if (tree.GetBTData<bool>(BTData.bReturning))
                return;
            if (isHeadshot) damage *= monsterStat.weeknessRatio;

            if (battleCoroutine != null) StopCoroutine(battleCoroutine);
            battleCoroutine = StartCoroutine(EndBattleState());

            currentHP -= damage;
            tree.SetBTData<float>(BTData.iCurrentHP, currentHP);

            if (currentHP < 1)
            {
                SetMonsterDead();
                HideBillboard();
                isBillboardOn = false;
            }
            else
            {
                if (isHeadshot)
                    audioController.PlayAudio(MonsterAudioType.HeadShot);
                else audioController.PlayAudio(MonsterAudioType.Hit);
            }

            isHeadshot = false;
        }

        // >> : Billboard
        private IEnumerator EndBattleState()
        {
            yield return new WaitForSeconds(MonsterCenter.battleStateTime);
            HideBillboard();
            isBillboardOn = false;
        }
        private void MonsterOnPlayerForward()
        {
            if (!isBillboardOn) return;

            Vector3 direction = transform.position - cameraObj.position;
            float dot = Vector3.Dot(direction.normalized, cameraObj.forward.normalized);

            if (dot > 0)
                ShowBillboard();
            else
                HideBillboard();
        }
        private void HideBillboard()
        {
            billboardObject.transform.localScale = Vector3.zero;
        }
        private void ShowBillboard()
        {
            billboardObject.transform.localEulerAngles = Vector3.one;
            isBillboardOn = true;
        }
    }
}
```

2-2.2 Monsters

- 평범한 스켈레톤 (NormalSkeleton)

```
using System.Collections;
using Assets.Scripts.Managers;

namespace Scripts.BehaviourTrees.Monster
{
    public class NormalSkeleton : MonsterBase
    {
        private enum ParsingData
        {
            MonsterStat = 1900,
            MeleeAttack = 2000,
            RunToPlayer = 2005,
        }

        private MeleeAttackData meleeAttackData;
        private RunToPlayerData runData;

        protected override void Start()
        {
            base.Start();
            StartCoroutine(InitParsingData());
        }
        private IEnumerator InitParsingData()
        {
            yield return DataManager.Instance.CheckIsParseDone();

            monsterStat = DataManager.Instance.GetIndexData<MonsterStatData,
MonsterStatDataParsingInfo>((int)ParsingData.MonsterStat);
            meleeAttackData = DataManager.Instance.GetIndexData<MeleeAttackData,
MonsterAttackDataparsingInfo>((int)ParsingData.MeleeAttack);
            runData = DataManager.Instance.GetIndexData<RunToPlayerData,
MonsterAttackDataparsingInfo>((int)ParsingData.RunToPlayer);

            SetSkills();
            SetMonsterData(monsterStat);

            tree.AddMonsterData<MonsterStatData>(MonsterData.MonsterStat, monsterStat);
            tree.AddMonsterData<MeleeAttackData>(MonsterData.Melee, meleeAttackData);
            tree.AddMonsterData<RunToPlayerData>(MonsterData.RunToPlayer, runData);

        }
        private void SetSkills()
        {
            MonsterMeleeAttack meleeAttack = new();
            meleeAttack.SetInitialData(meleeAttackData);
            AddAttack(MonsterSkill.Melee, meleeAttack);
        }
    }
}
```

- 모험가 스켈레톤 (AdventureSkeleton)

```
using System.Collections;
using Assets.Scripts.Managers;

namespace Scripts.BehaviourTrees.Monster
{
    public class AdventureSkeleton : MonsterBase
    {
        private enum ParsingData
        {
            MonsterStat = 1901,
            MeleeAttack = 2001,
            RunToPlayer = 2006,
```

```
            WeaponAttack = 2100,
        }

        private MeleeAttackData meleeAttackData;
        private RunToPlayerData runData;
        private WeaponAttackData weaponAttackData;

        protected override void Start()
        {
            base.Start();
            StartCoroutine(InitParsingData());
        }
        private IEnumerator InitParsingData()
        {
            yield return DataManager.Instance.CheckIsParseDone();

            monsterStat = DataManager.Instance.GetIndexData<MonsterStatData,
MonsterStatDataParsingInfo>((int)ParsingData.MonsterStat);
            meleeAttackData = DataManager.Instance.GetIndexData<MeleeAttackData,
MonsterAttackDataparsingInfo>((int)ParsingData.MeleeAttack);
            runData = DataManager.Instance.GetIndexData<RunToPlayerData,
MonsterAttackDataparsingInfo>((int)ParsingData.RunToPlayer);
            weaponAttackData = DataManager.Instance.GetIndexData<WeaponAttackData,
MonsterAttackDataparsingInfo>((int)ParsingData.WeaponAttack);

            SetSkills();
            SetMonsterData(monsterStat);

            tree.AddMonsterData<MonsterStatData>(MonsterData.MonsterStat, monsterStat);
            tree.AddMonsterData<MeleeAttackData>(MonsterData.Melee, meleeAttackData);
            tree.AddMonsterData<RunToPlayerData>(MonsterData.RunToPlayer, runData);
            tree.AddMonsterData<WeaponAttackData>(MonsterData.Weapon, weaponAttackData);
        }
        private void SetSkills()
        {
            MonsterMeleeAttack meleeAttack = new();
            meleeAttack.SetInitialData(meleeAttackData);
            AddAttack(MonsterSkill.Melee, meleeAttack);

            MonsterWeaponeAttack weaponeAttack = new();
            weaponeAttack.SetInitialData(weaponAttackData);
            AddAttack(MonsterSkill.Weapon, weaponeAttack);
        }

    }
}
```

- 마법사 스켈레톤 (WizardSkeleton)

```
using System.Collections;
using Assets.Scripts.Managers;

namespace Scripts.BehaviourTrees.Monster
{
    public class WizardSkeleton : MonsterBase
    {
        private enum ParsingData
        {
            MonsterStat = 1902,
            Flee = 2008,
            ProjectileAttack = 2009,
        }

        private FleeSkillData fleeData;
        private ProjectileAttackData projectileData;

        protected override void Start()
        {
```

```
            base.Start();
            StartCoroutine(InitParsingData());
        }
        private IEnumerator InitParsingData()
        {
            yield return DataManager.Instance.CheckIsParseDone();

            monsterStat = DataManager.Instance.GetIndexData<MonsterStatData,
MonsterStatDataParsingInfo>((int)ParsingData.MonsterStat);
            fleeData = DataManager.Instance.GetIndexData<FleeSkillData,
MonsterAttackDataparsingInfo>((int)ParsingData.Flee);
            projectileData = DataManager.Instance.GetIndexData<ProjectileAttackData,
MonsterAttackDataparsingInfo>((int)ParsingData.ProjectileAttack);

            SetSkills();
            SetMonsterData(monsterStat);

            tree.AddMonsterData<MonsterStatData>(MonsterData.MonsterStat, monsterStat);
            tree.AddMonsterData<FleeSkillData>(MonsterData.Melee, fleeData);
            tree.AddMonsterData<ProjectileAttackData>(MonsterData.RunToPlayer, projectileData);
        }

        private void SetSkills()
        {
            MonsterFleeSkill fleeAttack = new();
            fleeAttack.SetInitialData(monsterStat, fleeData);
            AddAttack(MonsterSkill.Flee, fleeAttack);

            MonsterProjectileAttack projectileAttack = new();
            projectileAttack.SetInitialData(projectileData);
            AddAttack(MonsterSkill.Projectile, projectileAttack);
        }
    }
}
```

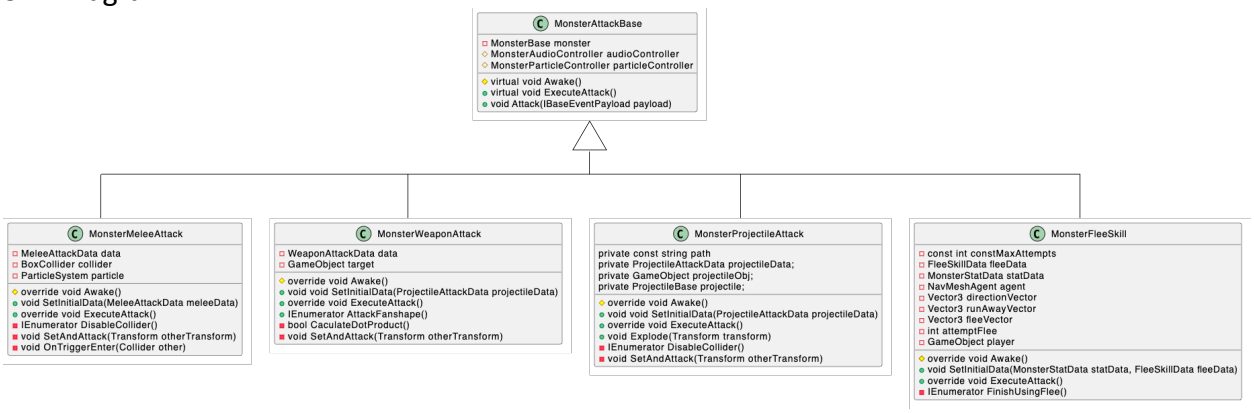
- 공대원 스켈레톤

```
using System.Collections;
using Assets.Scripts.Managers;

namespace Scripts.BehaviourTrees.Monster
{
    public class GuildguardSkeleton : MonsterBase
    {
        private enum ParsingData
        {
            MonsterStat = 1903,
            MeleeAttack = 2002,
            RunToPlayer = 2007,
            WeaponAttack = 2101,
        }

        private MeleeAttackData meleeAttackData;
        private RunToPlayerData runData;
        private WeaponAttackData weaponAttackData;

        protected override void Start()
        {
            base.Start();
            StartCoroutine(InitParsingData());
        }
        private IEnumerator InitParsingData()
        {
            yield return DataManager.Instance.CheckIsParseDone();

            monsterStat = DataManager.Instance.GetIndexData<MonsterStatData,
MonsterStatDataParsingInfo>((int)ParsingData.MonsterStat);
```

```
            meleeAttackData = DataManager.Instance.GetIndexData<MeleeAttackData,
MonsterAttackDataparsingInfo>((int)ParsingData.MeleeAttack);
            runData = DataManager.Instance.GetIndexData<RunToPlayerData,
MonsterAttackDataparsingInfo>((int)ParsingData.RunToPlayer);
            weaponAttackData = DataManager.Instance.GetIndexData<WeaponAttackData,
MonsterAttackDataparsingInfo>((int)ParsingData.WeaponAttack);

            SetSkills();
            SetMonsterData(monsterStat);

            tree.AddMonsterData<MonsterStatData>(MonsterData.MonsterStat, monsterStat);
            tree.AddMonsterData<MeleeAttackData>(MonsterData.Melee, meleeAttackData);
            tree.AddMonsterData<RunToPlayerData>(MonsterData.RunToPlayer, runData);
            tree.AddMonsterData<WeaponAttackData>(MonsterData.Weapon, weaponAttackData);
        }
        private void SetSkills()
        {
            MonsterMeleeAttack meleeAttack = new();
            meleeAttack.SetInitialData(meleeAttackData);
            AddAttack(MonsterSkill.Melee, meleeAttack);

            MonsterWeaponeAttack weaponeAttack = new();
            weaponeAttack.SetInitialData(weaponAttackData);
            AddAttack(MonsterSkill.Weapon, weaponeAttack);
        }
    }
}
```

3. 몬스터 공격

3-1. Diagram



3-2. Monster Attacks

3-2.1 MonsterAttackBase

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterAttackBase : MonoBehaviour
    {
        private MonsterBase monster;
        protected MonsterAudioController audioController;
        protected MonsterParticleController particleController;

        protected virtual void Awake()
        {
            if (audioController == null)
                audioController = GetComponent<MonsterAudioController>();
            if (particleController == null)
                particleController = GetComponent<MonsterParticleController>();
```

```
        }

        public virtual void ExecuteAttack() {}

        public void Attack(IBaseEventPayload payload)
        {
            monster.Attack(payload);
        }
    }
}
```

3-2.2 Monster Skills

- MonsterMeleeAttack (BoxcolliderAttack)

```
using System.Collections;
using Assets.Scripts.Combat;
using Assets.Scripts.StatusEffects;
using Channels.Combat;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterMeleeAttack : MonsterAttackBase
    {
        private MeleeAttackData data;
        private BoxCollider collider;
        private ParticleSystem particle;

        protected override void Awake()
        {
            base.Awake();

            // Collider Setting
            if (collider == null)
                collider = gameObject.AddComponent<BoxCollider>();
            collider.isTrigger = true;
            collider.size = data.colliderSize;
            gameObject.transform.localPosition = data.colliderOffset;
            collider.enabled = false;
        }
        public void SetInitialData(MeleeAttackData meleeData)
        {
            data = meleeData;
        }
        public override void ExecuteAttack()
        {
            if(data==null)
                Debug.LogFormat("{0} Has Not Initialized MeleeAttack : {0}, MeleeAttack" +
transform.name);
            collider.enabled = true;
            StartCoroutine(DisableCollider());
        }
        private IEnumerator DisableCollider()
        {
            yield return new WaitForSeconds(data.colliderDuration);
            collider.enabled = false;
        }
        private void OnTriggerEnter(Collider other)
        {
            if(other.CompareTag("Player"))
            {
                if(other.gameObject.GetComponent<ICombatant>()!=null)
                {
                    audioController.PlayAudio(MonsterAudioType.MeleeAttackHit);
                    if (particle == null)
                    {
                        particle = particleController.GetParticle(MonsterParticleType.MeleeHit);
                    }
```

```
                    particle.transform.position = other.transform.position;
                    particle.Play();
                    SetAndAttack(other.transform);
                }
            }
        }
        private void SetAndAttack(Transform otherTransform)
        {
            CombatPayload payload = new();
            payload.Type = data.combatType;
            payload.Attacker = transform;
            payload.Defender = otherTransform;
            payload.AttackDirection = Vector3.zero;
            payload.AttackStartPosition = transform.position;
            payload.AttackPosition = otherTransform.position;
            payload.StatusEffectName = StatusEffectName.WeakRigidity;
            payload.statusEffectduration = 0.3f;
            payload.Damage = data.damage;
            Attack(payload);
        }
    }
}
```

- MonsterWeaponAttack (내적 계산 공격)

```
using System.Collections;
using Assets.Scripts.StatusEffects;
using Channels.Combat;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterWeaponeAttack : MonsterAttackBase
    {
        private WeaponAttackData data;
        private GameObject target;

        protected override void Awake()
        {
            base.Awake();
            target = GameObject.Find("Player");
        }
        public void SetInitialData(WeaponAttackData WeaponData)
        {
            data = WeaponData;
        }
        public override void ExecuteAttack()
        {
            StartCoroutine(AttackFanshape());
        }
        public IEnumerator AttackFanshape()
        {
            float accumTime = 0.0f;
            while (accumTime <= data.duration)
            {
                if (CaculateDotProduct())
                {
                    if (target.CompareTag("Player"))
                    {
                        audioController.PlayAudio(MonsterAudioType.WeaponAttackHit);
                        particleController.PlayParticle(MonsterParticleType.WeaponHit,
target.transform);

                        SetAndAttack(data, target.transform);
                        break;
                    }
                }
                accumTime += Time.deltaTime;
```

```
                yield return null;
            }
        }
        private bool CaculateDotProduct()
        {
            Vector3 interV = target.transform.position - transform.position;

            float dot = Vector3.Dot(interV.normalized, transform.forward.normalized);
            float theta = Mathf.Acos(dot);
            float degree = Mathf.Rad2Deg * theta;

            if (degree <= data.angle / 2.0f)
            {
                interV.y = 0;
                if (interV.sqrMagnitude <= data.radius * data.radius)
                {
                    return true;
                }
            }

            return false;
        }
        private void SetAndAttack(WeaponAttackData data, Transform otherTransform)
        {
            CombatPayload payload = new();
            payload.Type = data.combatType;
            payload.Attacker = transform;
            payload.Defender = otherTransform;
            payload.AttackDirection = Vector3.zero;
            payload.AttackStartPosition = transform.position;
            payload.AttackPosition = otherTransform.position;
            payload.StatusEffectName = StatusEffectName.WeakRigidity;
            payload.statusEffectduration = 0.5f;
            payload.Damage = data.damage;
            Attack(payload);
        }
    }
}
```

- MonsterProjectileAttack(투사체 공격)

```
using Assets.Scripts.Managers;
using Assets.Scripts.StatusEffects;
using Channels.Combat;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterProjectileAttack : MonsterAttackBase
    {
        private const string path = "Prefabs/Monster/Projectiles/";

        private ProjectileAttackData projectileData;
        private GameObject projectileObj;
        private ProjectileBase projectile;

        protected override void Awake()
        {
            base.Awake();

            string finalPath = path + projectileData.name;
            projectileObj =
Instantiate(ResourceManager.Instance.LoadExternResource<GameObject>(finalPath), transform);
            if (projectileObj == null)
                Debug.LogFormat("{0} Trying To Instanciate {1} But No Prefab On {2} : {0}, {1}",
transform.name, projectileData.name, path);
            else
                projectile = projectileObj.GetComponent<ProjectileBase>();
```

```
        }
        public void SetInitialData(ProjectileAttackData projectileData)
        {
            this.projectileData = projectileData;
        }
        public override void ExecuteAttack()
        {
            if (projectileData == null)
                Debug.LogFormat("{0} Has Not Initialized {1} : {0}, {1}" + transform.name,
projectileData.name);
            else projectile.Fire();
        }
        public void Explode(Transform transform)
        {
            particleController.PlayParticle(MonsterParticleType.ProjectileHit, transform);
            audioController.PlayAudio(MonsterAudioType.ProjectileHit, transform);
        }
        public void SetAndAttack(Transform otherTransform)
        {
            CombatPayload payload = new();
            payload.Type = projectileData.combatType;
            payload.Attacker = transform;
            payload.Defender = otherTransform;
            payload.AttackDirection = Vector3.zero;
            payload.AttackStartPosition = transform.position;
            payload.AttackPosition = otherTransform.position;
            payload.StatusEffectName = StatusEffectName.Burn;
            payload.statusEffectduration = 3.0f;
            payload.Damage = (int)projectileData.damage;
            Attack(payload);
        }
    }
}
```

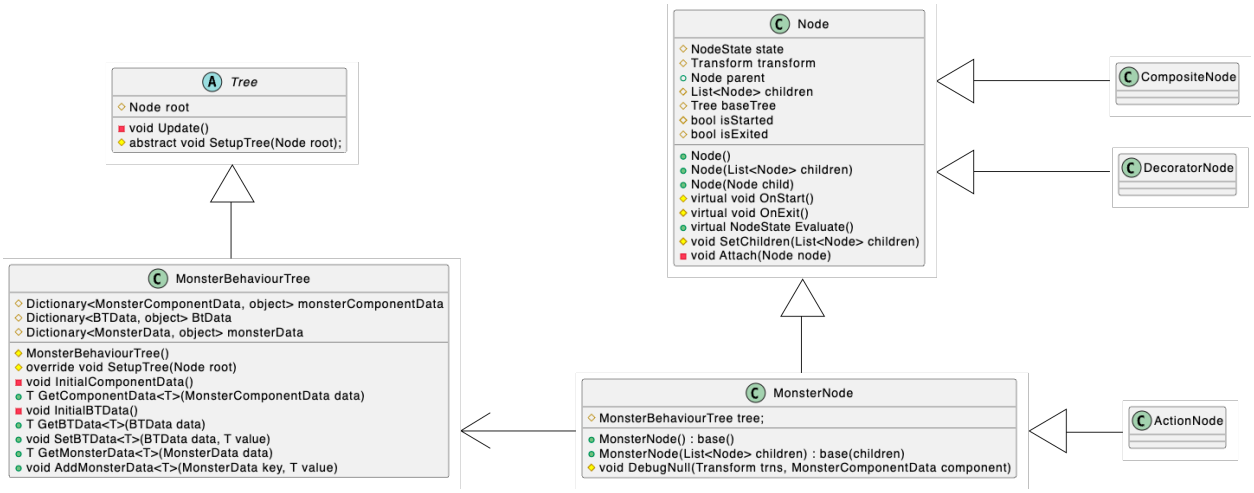
- MonsterFleeSkill (도망가기)

```
using System.Collections;
using UnityEngine;
using UnityEngine.AI;

namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterFleeSkill : MonsterAttackBase
    {
        private const int constMaxAttempts = 5;
        private FleeSkillData fleeData;
        private MonsterStatData statData;
        private NavMeshAgent agent;

        private Vector3 directionVector;
        private Vector3 runAwayVector;
        private Vector3 fleeVector;
        private int attemptFlee;
        private GameObject player;

        protected override void Awake()
        {
            base.Awake();
            player = GameObject.Find("Player");
            agent = GetComponent<NavMeshAgent>();

        }
        public void SetInitialData(MonsterStatData statData, FleeSkillData fleeData)
        {
            this.statData = statData;
            this.fleeData = fleeData;
        }
        public override void ExecuteAttack()
```

```
        {
            if(statData==null||fleeData==null)
            {
                Debug.LogFormat("{0} Has Not Initialized WeaponAttack : {0}, WeaponAttack" +
transform.name);
                return;
            }
            attemptFlee = 0;
            directionVector = player.transform.position - transform.position;
            directionVector.y = 0.0f;
            runAwayVector = directionVector.normalized * -fleeData.fleeDistance;
            fleeVector = transform.position + runAwayVector;

            // Check Walls
            RaycastHit hit;
            do
            {
                bool isHittedWall = true;
                if (Physics.Raycast
                    (transform.position, runAwayVector.normalized, out hit, fleeData.fleeDistance))
                {
                    if (hit.collider.tag == "Wall")
                    {
                        runAwayVector = Random.onUnitSphere;
                    }
                    else isHittedWall = false;
                }
                else isHittedWall = false;

                // Check NavMesh
                if (!isHittedWall)
                {
                    NavMeshHit navMeshHit;
                    if (NavMesh.SamplePosition(fleeVector, out navMeshHit, 1.0f, NavMesh.AllAreas))
                    {
                        // Found Runaway Point
                        agent.destination = navMeshHit.position;
                        agent.speed = fleeData.fleeSpeed;
                        StartCoroutine(FinishUsingFlee());
                        break;
                    }
                }

                attemptFlee++;
            } while (attemptFlee < constMaxAttempts);
        }
        private IEnumerator FinishUsingFlee()
        {
            yield return new WaitForSeconds(fleeData.duration - fleeData.animationHold);
            agent.speed = statData.speed;
        }
    }
}
```

4. Behaviour Tree

4-1. Behaviour Tree & Node



- Tree

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public abstract class Tree : MonoBehaviour
    {
        protected Node root = null;

        private void Update()
        {
            if (root != null) root.Evaluate();
        }
        protected abstract void SetupTree(Node root);
    }
}
```

- MonsterBehaviourTree

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterBehaviourTree : Tree
    {
        protected Dictionary<MonsterComponentData, object> monsterComponentData = new();
        protected Dictionary<BTData, object> BtData = new();
        protected Dictionary<MonsterData, object> monsterData;

        protected MonsterBehaviourTree()
        {
            InitialComponentData();
            InitialBTData();
        }
        protected override void SetupTree(Node root)
        {
            this.root = root;
        }
        // >> : ComponentData
        private void InitialComponentData()
        {
            NavMeshAgent agent = GetComponent<NavMeshAgent>();
```

```
            monsterComponentData.Add(MonsterComponentData.AGENT, agent);

            MonsterBase monster = GetComponent<MonsterBase>();
            monsterComponentData.Add(MonsterComponentData.MONSTER, monster);

            MonsterParticleController particle = GetComponent<MonsterParticleController>();
            monsterComponentData.Add(MonsterComponentData.PARTICLE, particle);

            AudioSource audioSource = GetComponent<AudioSource>();
            monsterComponentData.Add(MonsterComponentData.AUDIO, audioSource);

            MonsterAudioController audioController = GetComponent<MonsterAudioController>();
            monsterComponentData.Add(MonsterComponentData.AUDIO_CON, audioController);

            Transform transform = GetComponent<Transform>();
            monsterComponentData.Add(MonsterComponentData.TRANSFORM, transform);

            Animator animator = GetComponent<Animator>();
            monsterComponentData.Add(MonsterComponentData.ANIMATOR, animator);

            Transform patrolPoints = transform.Find("PatrolPoints");
            monsterComponentData.Add(MonsterComponentData.PATROL_POINTS, patrolPoints);

            DetectAI detectPlayer = transform.Find("DetectPlayerAI").GetComponent<DetectAI>();
            monsterComponentData.Add(MonsterComponentData.PlayerDetectAI, detectPlayer);

            DetectAI detectChase = transform.Find("DetectChaseAI").GetComponent<DetectAI>();
            monsterComponentData.Add(MonsterComponentData.ChaseDetectAI, detectChase);

            Vector3[] patorlPoints =
transform.Find("PatrolPoints").GetComponent<PatrolPoints>().GetPatrolPointst();
            monsterComponentData.Add(MonsterComponentData.PATROL_POINTS, patrolPoints);
        }
        public T GetComponentData<T>(MonsterComponentData data)
        {
            object obj = null;
            if (monsterComponentData.TryGetValue(data, out obj))
            {
                if (obj is T) return (T)obj;
            }
            else
            {
                Debug.LogFormat("Trying To Get Type Does Not Match : {0}, {1} " + data.ToString(),
transform.name);
                return default(T);
            }

            Debug.LogFormat(gameObject.name + "Trying To Access Object Does Not Have : {0}, {1}" +
data.ToString(), transform.name);
            return default(T);
        }

        // >> : BTData
        private void InitialBTData()
        {
            SetBTData<bool>(BTData.bOnSpawnPosition, true);
            SetBTData<bool>(BTData.bOvertraveld, false);
            SetBTData<bool>(BTData.bReturning, false);
        }
        public T GetBTData<T>(BTData data)
        {
            object obj;
            if (BtData.TryGetValue(data, out obj))
            {
                if (obj is T) return (T)obj;
                else
                {
                    Debug.LogFormat("Trying To Get Component Does Not Match : {0}, {1} " +
data.ToString(), transform.name);
```

```
                    return default(T);
                }
            }
            else
            {
                T defaultValue = default(T);
                BtData.Add(data, defaultValue);
                return defaultValue;
            }
        }
        public void SetBTData<T>(BTData data, T value)
        {
            if (BtData.ContainsKey(data))
            {
                if (BtData[data] is T)
                {
                    BtData[data] = value;
                    return;
                }
                else
                {
                    Debug.LogFormat("Trying To Get Data Does Not Match : {0}, {1} " +
data.ToString(), transform.name);
                    return;
                }
            }

            Debug.LogFormat("Trying to access data that does not exist: {0}, {1}", data.ToString(),
transform.name);
        }

        // >> : MonsterData
        public T GetMonsterData<T>(MonsterData data)
        {
            object obj;
            if (monsterData.TryGetValue(data, out obj))
            {
                if (obj is T) return (T)obj;
                else
                {
                    Debug.LogFormat("Trying To Get Monster Data Does Not Match : {0}, {1} " +
data.ToString(), transform.name);
                    return default(T);
                }
            }

            Debug.LogFormat(gameObject.name + "Trying To Access Monster Data Does Not Have : {0},
{1}" + data.ToString(), transform.name);
            return default(T);
        }
        public void AddMonsterData<T>(MonsterData key, T value)
        {
            if(monsterData.ContainsKey(key))
            {
                Debug.LogFormat("{0} Is Trying To Add Key {1} Already Exists : {0}, {1}",
transform.name, key.ToString());
                return;
            }
            monsterData.Add(key, value);
        }
    }
}
```

- Node

```
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
```

```
{
    public enum NodeState
    {
        RUNNING,
        SUCCESS,
        FAILURE,
    }

    public class Node
    {
        protected NodeState state;
        protected Transform transform;
        public Node parent;
        protected List<Node> children;
        protected Tree baseTree;
        protected bool isStarted;
        protected bool isExited;

        public Node()
        {
            parent = null;
        }
        public Node(List<Node> children)
        {
            foreach (Node child in children)
                Attach(child);
        }
        public Node(Node child)
        {
            Attach(child);
        }

        protected virtual void OnStart() { }
        protected virtual void OnExit() { }
        public virtual NodeState Evaluate() => NodeState.FAILURE;

        protected void SetChildren(List<Node> children)
        {
            foreach (Node child in children)
                Attach(child);
        }

        private void Attach(Node node)
        {
            node.parent = this;
            children.Add(node);
        }
    }
}
```

- MonsterNode

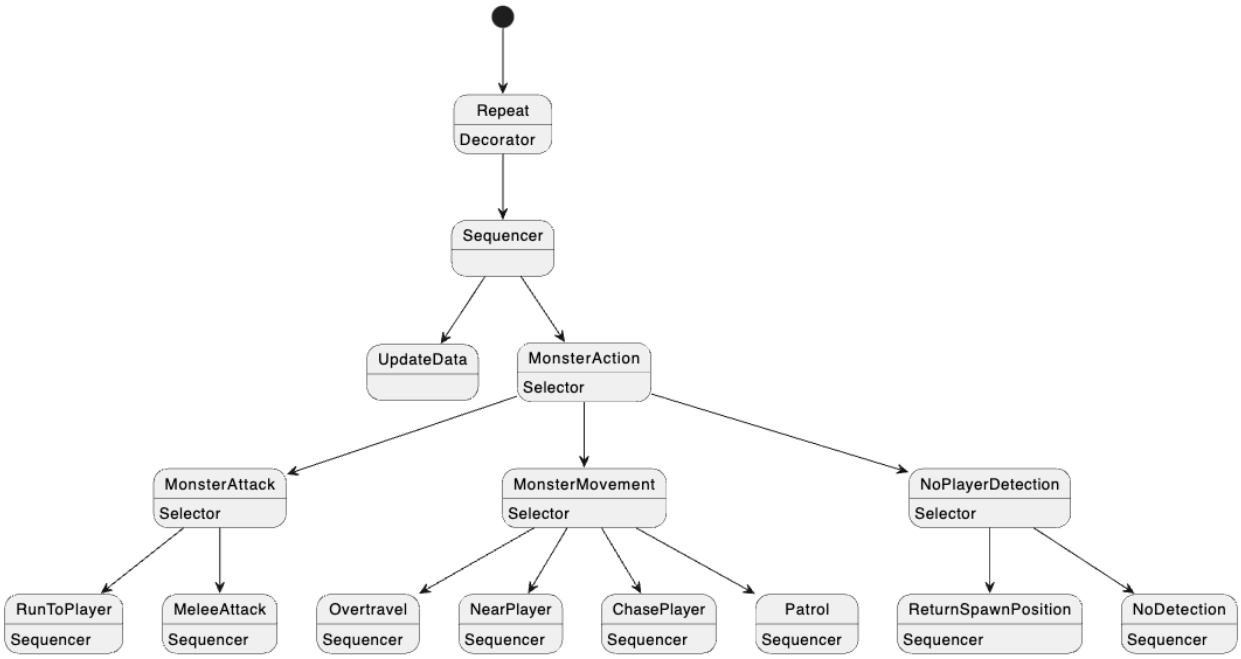
```
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterNode : Node
    {
        protected MonsterBehaviourTree tree;

        public MonsterNode() : base()
        {
            tree = baseTree as MonsterBehaviourTree;
            if (tree == null)
                Debug.LogFormat("Tree DownCasting To MonsterBehaviourTree Failed : {0}",
transform.name);
            transform = tree.GetComponentData<Transform>(MonsterComponentData.TRANSFORM);
        }
```

```
        public MonsterNode(List<Node> children) : base(children) { }

        protected void DebugNull(Transform trns, MonsterComponentData component)
        {
            Debug.LogFormat("{0} Is Trying To Access {1}, Which It Does Not Have : {0} / {1}",
trns.name, component.ToString());
        }
    }
}
```

4-2. Monster Behaviour Tree

4-2.1 평범한 스켈레톤

- 설계



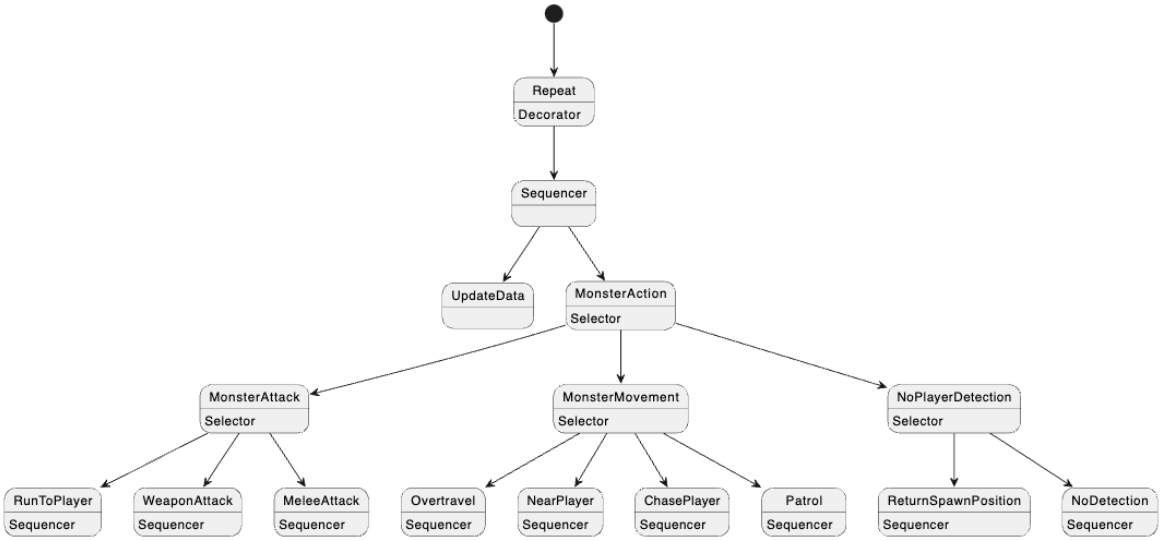
- 코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class NormalSkeletonBT : MonsterBehaviourTree
    {
        private void Start()
        {
            // Monster Attack Sequences
            Select attackSelect = new(new List<Node>
            {
                new SeqRunToPlayer(),
                new SeqMeleeAttack()
            });

            // Monster Cant Attack, But Can Move Sequences
            Select movementSelect = new(new List<Node>
            {
                new SeqOvertravel(),
                new SeqNearPlayer(),
                new SeqChasePlayer(),
                new SeqPatrol(),
```

```
            });

            // Monster Cant detect player Sequences
            Select noDetectSelect = new(new List<Node>
            {
                new SeqReturnSpawnPosition(),
                new SeqNoDetection(),
            });

            // Combine All Sequences
            Select allSequence = new(new List<Node>
            {
                attackSelect,
                movementSelect,
                noDetectSelect,
            });

            // Combine Update Data
            Sequence normalSkeletonNodes = new(new List<Node>
            {
                new ActionUpdateData(),
                allSequence,
            });

            SetupTree(new Repeater(normalSkeletonNodes));
        }
    }
}
```

4-2.2 모험가 스켈레톤

- 설계



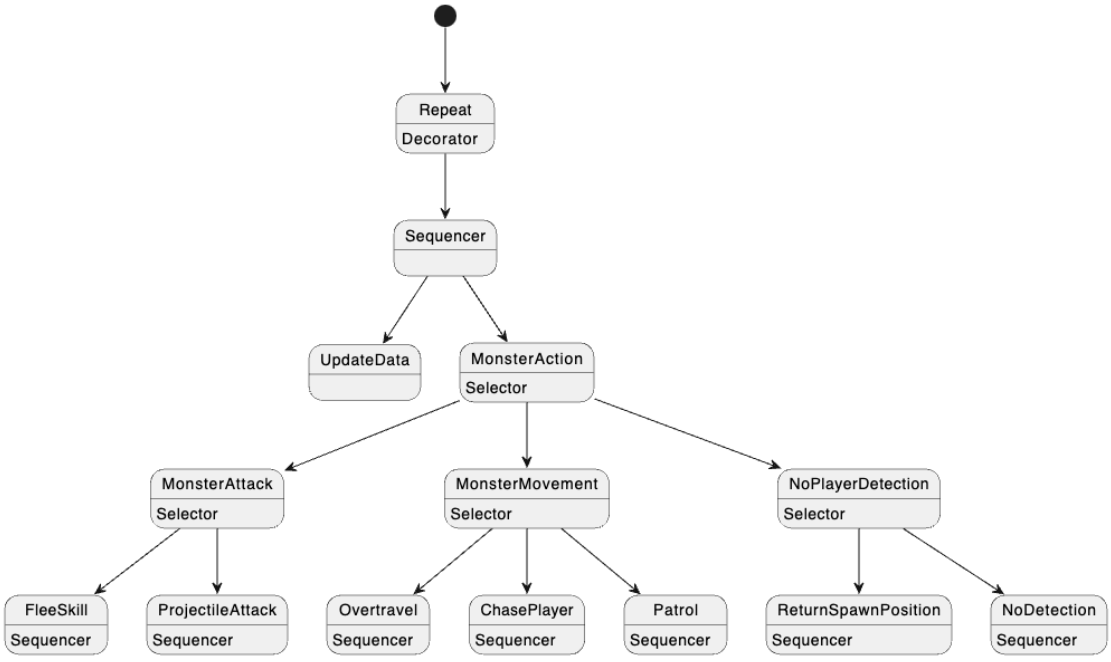
- 코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class AdventureSkeletonBT : MonsterBehaviourTree
    {
        private void Start()
        {
```

```
            Select attackSelect = new(new List<Node>
            {
                new SeqRunToPlayer(),
                new SeqWeaponAttack(),
                new SeqMeleeAttack()
            });
            Select movementSelect = new(new List<Node>
            {
                new SeqOvertravel(),
                new SeqNearPlayer(),
                new SeqChasePlayer(),
                new SeqPatrol(),
            });
            Select noDetectSelect = new(new List<Node>
            {
                new SeqReturnSpawnPosition(),
                new SeqNoDetection(),
            });

            Select allSequence = new(new List<Node>
            {
                attackSelect,
                movementSelect,
                noDetectSelect,
            });

            Sequence adventureSkeletonNodes = new(new List<Node>
            {
                new ActionUpdateData(),
                allSequence,
            });

            SetupTree(new Repeater(adventureSkeletonNodes));
        }
    }
}
```

4-2.3 마법사 스켈레톤

- 설계

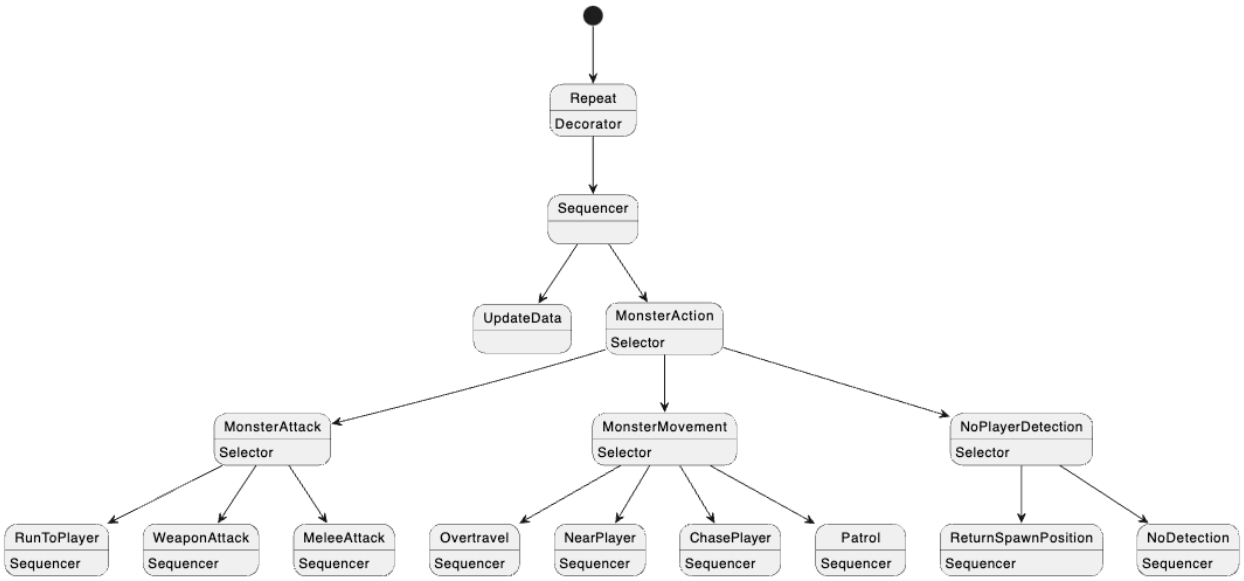


- 코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class WizardSkeletonBT : MonsterBehaviourTree
    {
        private void Start()
        {
            Select attackSelect = new(new List<Node>
            {
                new SeqFleeSkill(),
                new SeqProjectileAttack(),
            });
            Select movementSelect = new(new List<Node>
            {
                new SeqOvertravel(),
                new SeqChasePlayer(),
                new SeqPatrol(),
            });
            Select noDetectSelect = new(new List<Node>
            {
                new SeqReturnSpawnPosition(),
                new SeqNoDetection(),
            });

            Select allSequence = new(new List<Node>
            {
                attackSelect,
                movementSelect,
                noDetectSelect,
            });

            Sequence wizardSkeletonNodes = new(new List<Node>
            {
                new ActionUpdateData(),
                allSequence,
            });

            SetupTree(new Repeater(wizardSkeletonNodes));
        }
    }
}
}
```

4-2.4 공대원 스켈레톤

- 설계



- 코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class GuildGuardSkeletonBT : MonsterBehaviourTree
    {
        private void Start()
        {
            Select attackSelect = new(new List<Node>
            {
                new SeqRunToPlayer(),
                new SeqWeaponAttack(),
                new SeqMeleeAttack()
            });
            Select movementSelect = new(new List<Node>
            {
                new SeqOvertravel(),
                new SeqNearPlayer(),
                new SeqChasePlayer(),
                new SeqPatrol(),
            });
            Select noDetectSelect = new(new List<Node>
            {
                new SeqReturnSpawnPosition(),
                new SeqNoDetection(),
            });

            Sequence allSequence = new(new List<Node>
            {
                attackSelect,
                movementSelect,
                noDetectSelect,
            });

            Sequence guildguardSkeletonNodes = new(new List<Node>
            {
                new ActionUpdateData(),
```

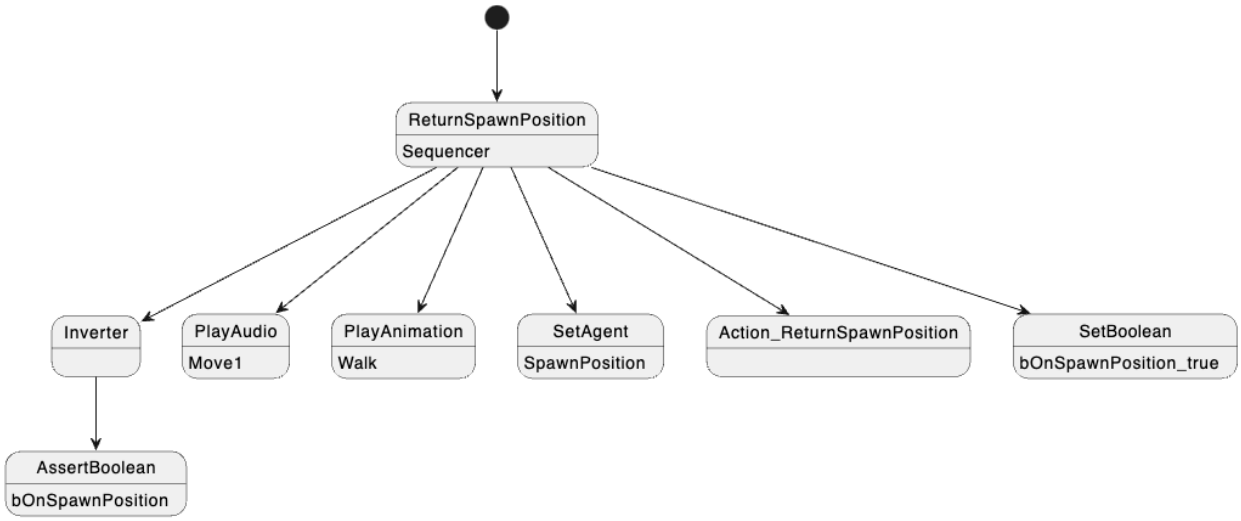
```
                allSequence,
            });

            SetupTree(new Repeater(guildguardSkeletonNodes));
        }
    }
}
```

4-3. Sequence Behaviour Tree

4-3.1 플레이어 미감지

- ReturnSpawnPosition

설계

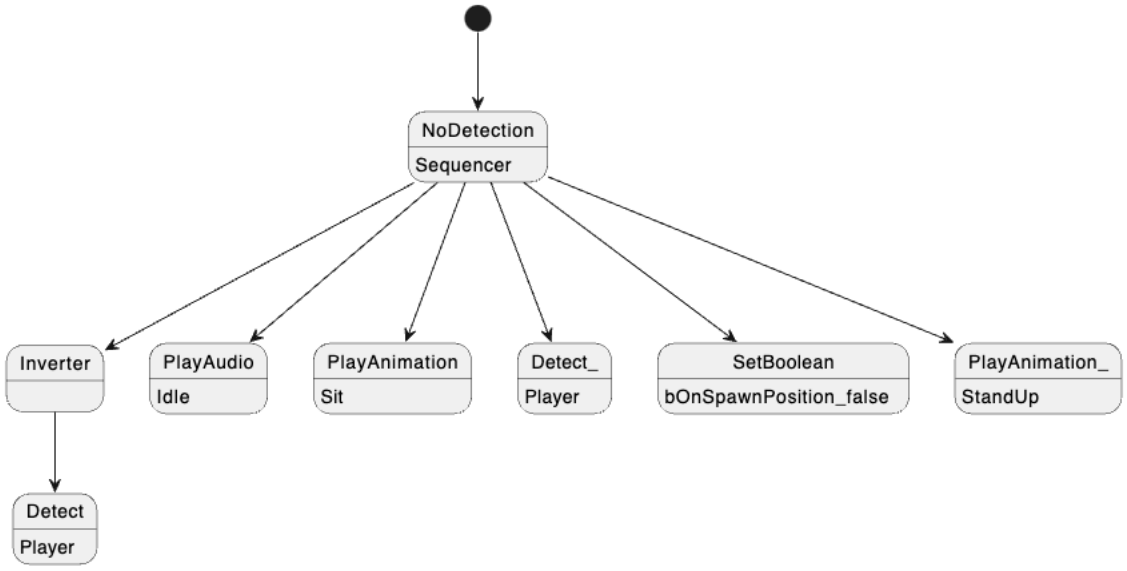


코드

```
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqReturnSpawnPosition : Sequence
    {
        public SeqReturnSpawnPosition()
        {
            List<Node> children = new()
            {
               new Inverter(new ActionAssertBoolean(tree.GetBTData<bool>(BTData.bOnSpawnPosition))),
               new ActionPlayAudio(MonsterAudioType.Move1, true, true),
               new ActionPlayAnimation(AnimationType.WALK),
               new ActionSetAgent(tree.GetMonsterData<Vector3>(MonsterData.v3SpawnPosition)),
               new ActionReturnSpawnPosition(),
               new ActionSetBoolean(BTData.bOnSpawnPosition, true),
            };

            SetChildren(children);
        }
    }
}
```

- NoDetection

설계



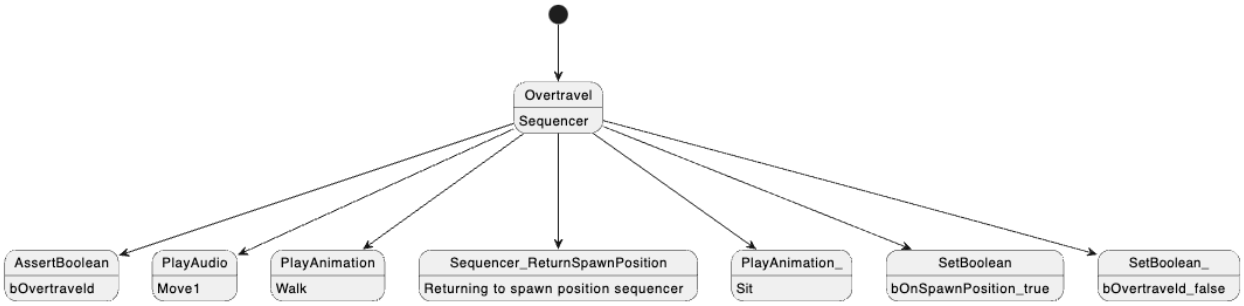
코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqNoDetection : Sequence
    {
        public SeqNoDetection()
        {
            List<Node> children = new()
            {
                // undetected player
                new Inverter(new ActionDetect(DetectType.PLAYER)),
                new ActionPlayAudio(MonsterAudioType.Idle,false, true),
                new ActionPlayAnimation(AnimationType.SIT, true),
                new ActionDetect(DetectType.PLAYER),
                // if detected player
                new ActionSetBoolean(BTData.bOnSpawnPosition, false),
                new ActionPlayAnimation(AnimationType.STANDUP, true),
            };

            SetChildren(children);
        }
    }
}
```

4-3.2 움직임

- Overtravel

설계

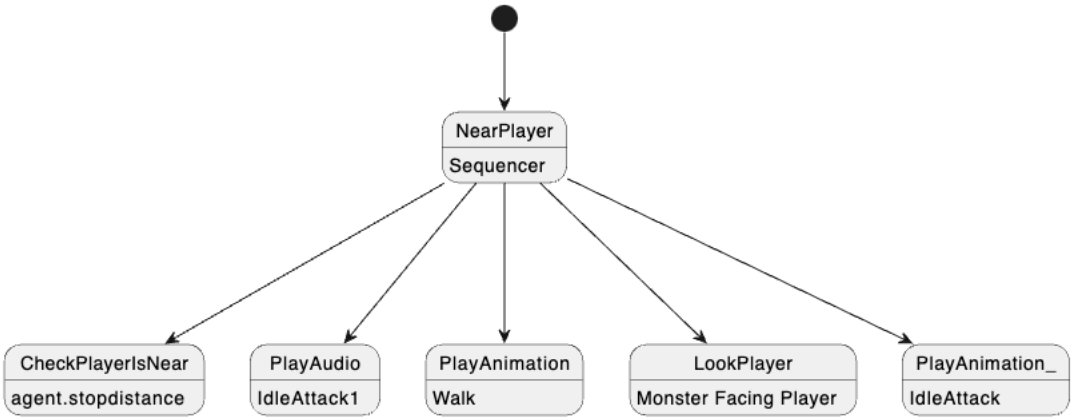


코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqOvertravel : Sequence
    {
        public SeqOvertravel()
        {
            List<Node> children = new()
            {
                new ActionAssertBoolean(tree.GetBTData<bool>(BTData.bOvertraveld)),
                new ActionPlayAudio(MonsterAudioType.Move1),
                new ActionPlayAnimation(AnimationType.WALK),
                new SeqReturnSpawnPosition(),
                new ActionPlayAnimation(AnimationType.SIT),
                new ActionSetBoolean(BTData.bOnSpawnPosition, true),
                new ActionSetBoolean(BTData.bOvertraveld, false),
            };

            SetChildren(children);
        }
    }
}
```

- NearPlayer

설계

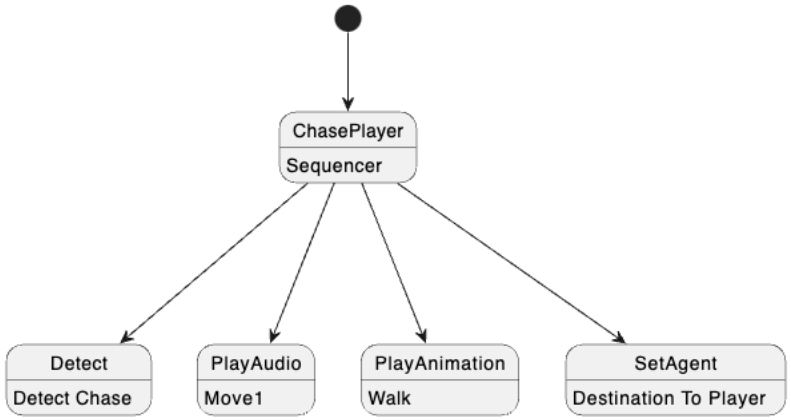


코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqNearPlayer : Sequence
    {
        public SeqNearPlayer()
        {
            List<Node> children = new()
            {
                new ActionCheckPlayerIsNear(),
                new ActionPlayAudio(MonsterAudioType.IdleAttack1),
                new ActionPlayAnimation(AnimationType.WALK),
                new ActionLookPlayer(),
                new ActionPlayAnimation(AnimationType.IDLE_ATTACK),
            };

            SetChildren(children);
        }
    }
}
```

- ChasePlayer

설계



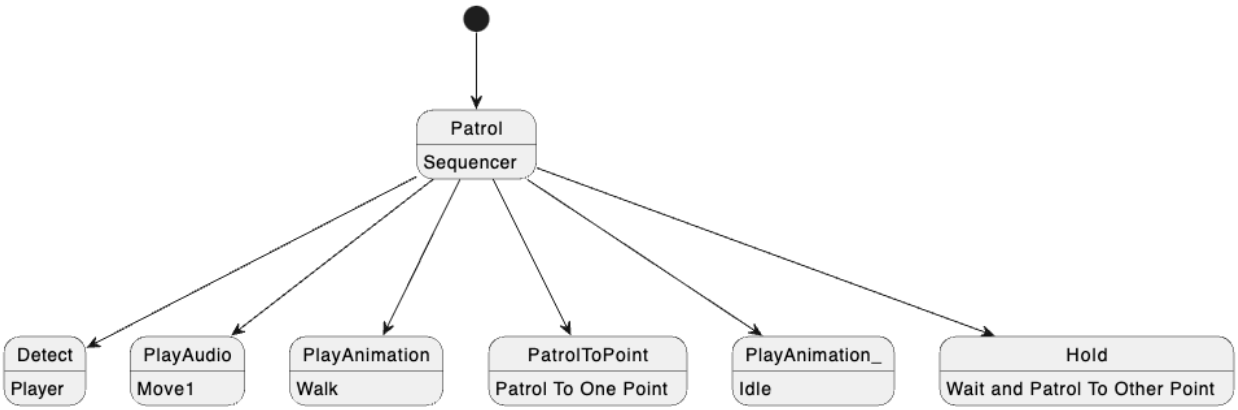
코드

```
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqChasePlayer : Sequence
    {
        public SeqChasePlayer()
        {
            List<Node> children = new()
            {
            new ActionDetect(DetectType.CHASE),
            new ActionPlayAudio(MonsterAudioType.Move1),
            new ActionPlayAnimation(AnimationType.WALK),
            new ActionSetAgent(tree.GetBTData<Vector3>(BTData.v3PlayerPosition)),
            };

            SetChildren(children);
        }
```

```
    }
}
```

Patrol

설계



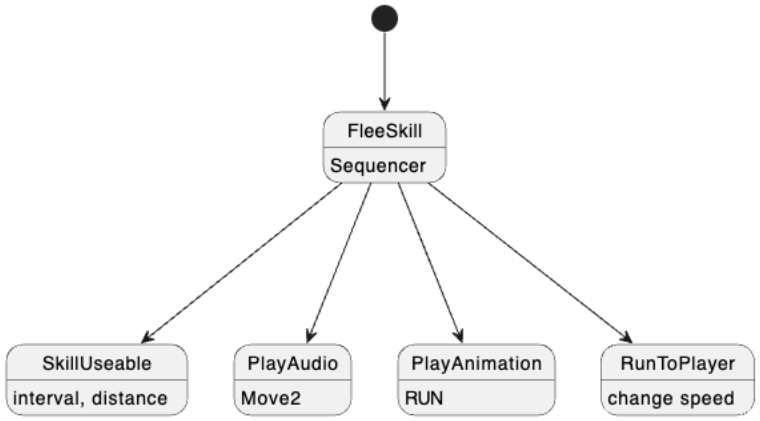
코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqPatrol : Sequence
    {
        public SeqPatrol()
        {
            List<Node> children = new()
            {
                new ActionDetect(DetectType.PLAYER),
                new ActionPlayAudio(MonsterAudioType.Move1),
                new ActionPlayAnimation(AnimationType.WALK),
                new ActionPatrolToPoint(),
                new ActionPlayAnimation(AnimationType.IDLE),
                new ActionHold(1.0f),
            };

            SetChildren(children);
        }

    }
}
```

4-3.3 공격

- RunToPlayer

설계

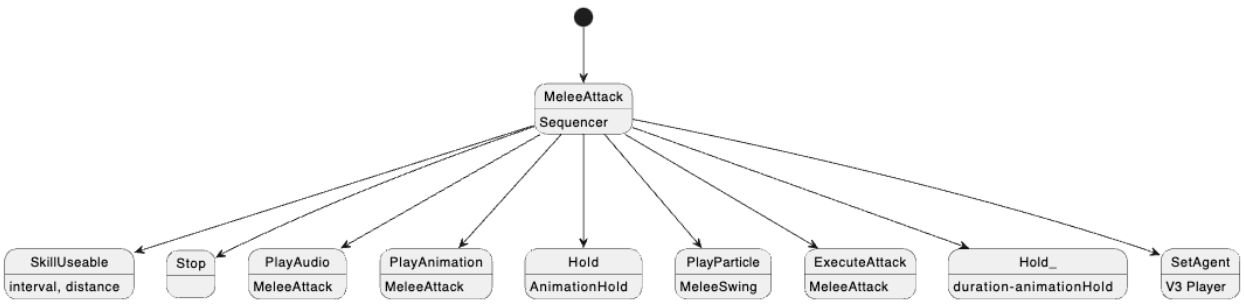


코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqRunToPlayer : Sequence
    {
        public SeqRunToPlayer()
        {
            RunToPlayerData runToPlayer =
tree.GetMonsterData<RunToPlayerData>(MonsterData.RunToPlayer);

            List<Node> children = new()
            {
                new ActionSkillUseable(runToPlayer.interval, runToPlayer.attackableDistance),
                new ActionPlayAudio(MonsterAudioType.MoveSkill),
                new ActionPlayAnimation(AnimationType.RUN),
                new ActionRunToPlayer()
            };

            SetChildren(children);
        }
    }
}
```

- MeleeAttack

설계

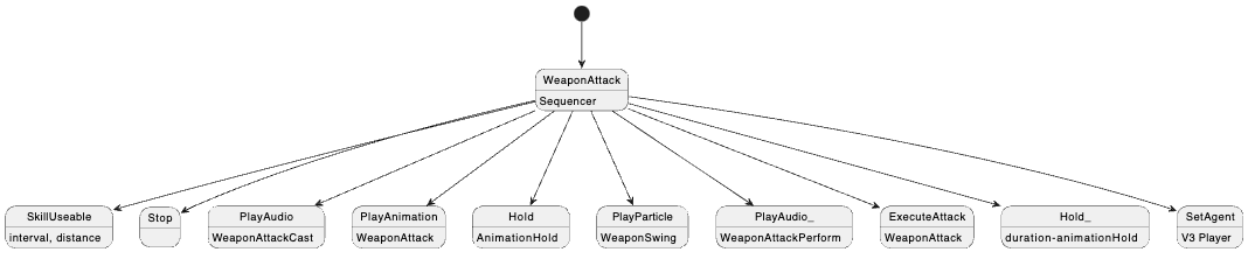


코드

```
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqMeleeAttack : Sequence
    {
        public SeqMeleeAttack()
        {
            MeleeAttackData meleeAttack = tree.GetMonsterData<MeleeAttackData>(MonsterData.Melee);

            List<Node> children = new()
            {
                new ActionSkillUseable(meleeAttack.interval, meleeAttack.attackableDistance),
                new ActionStop(),
                new ActionPlayAudio(MonsterAudioType.MeleeAttack,true, false),
                new ActionPlayAnimation(AnimationType.MELEE),
                new ActionHold(meleeAttack.animationHold),
                new ActionPlayParticle(MonsterParticleType.MeleeAttack),
                new ActionExecuteAttack(MonsterSkill.Melee),
                new ActionHold(meleeAttack.duration-meleeAttack.animationHold),
                new ActionSetAgent(tree.GetBTData<Vector3>(BTData.v3PlayerPosition)),
            };

            SetChildren(children);
        }
    }
}
```

- WeaponAttack

설계



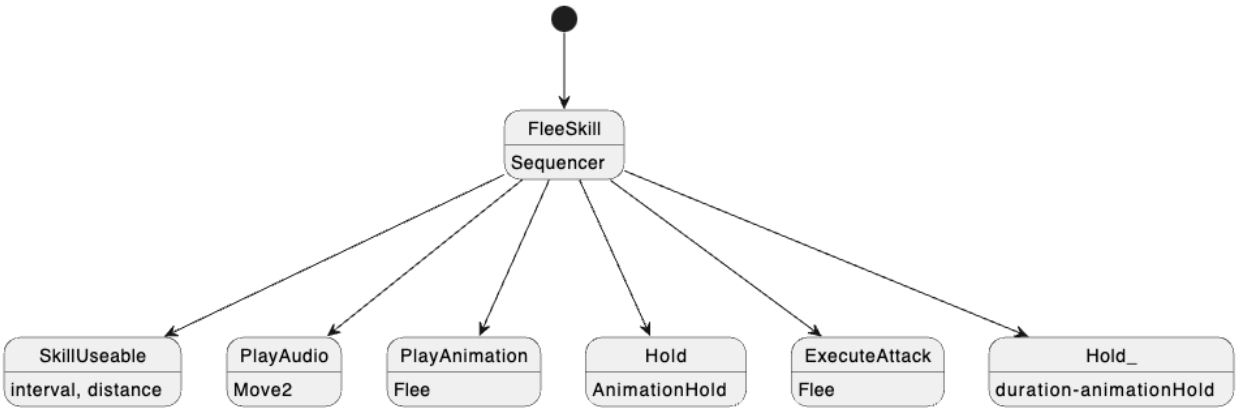
코드

```
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqWeaponAttack : Sequence
    {
        public SeqWeaponAttack()
        {
            WeaponAttackData weaponData =
tree.GetMonsterData<WeaponAttackData>(MonsterData.Weapon);

            List<Node> children = new()
```

```
            {
                new ActionSkillUseable(weaponData.interval,
weaponData.attackableDistance),
                new ActionStop(),
                new ActionPlayAudio(MonsterAudioType.WeaponAttackCast),
                new ActionPlayAnimation(AnimationType.WEAPON),
                new ActionHold(weaponData.animationHold),
                new ActionPlayParticle(MonsterParticleType.WeaponSwing),
                new ActionPlayAudio(MonsterAudioType.WeaponAttackPerform),
                new ActionExecuteAttack(MonsterSkill.Weapon),
                new ActionHold(weaponData.duration-weaponData.animationHold),
                new
ActionSetAgent(tree.GetBTData<Vector3>(BTData.v3PlayerPosition)),
            };

            SetChildren(children);
        }

    }
}
```

- FleeSkill

설계



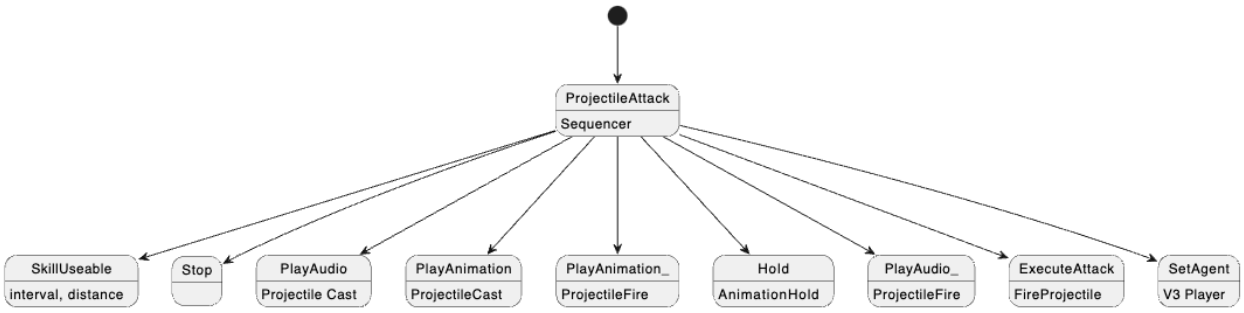
코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqFleeSkill : Sequence
    {
        public SeqFleeSkill()
        {
            FleeSkillData fleeData = tree.GetMonsterData<FleeSkillData>(MonsterData.Flee);

            List<Node> children = new()
            {
                new ActionSkillUseable(fleeData.interval, fleeData.attackableDistance),
                new ActionPlayAudio(MonsterAudioType.Move2),
                new ActionPlayAnimation(AnimationType.FLEE),
```

```
                new ActionHold(fleeData.animationHold),
                new ActionExecuteAttack(MonsterSkill.Flee),
                new ActionHold(fleeData.duration-fleeData.animationHold),
            };

            SetChildren(children);
        }
    }
}
```

- ProjectileAttack

설계



코드

```
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqProjectileAttack : Sequence
    {
        public SeqProjectileAttack()
        {
            ProjectileAttackData data =
tree.GetMonsterData<ProjectileAttackData>(MonsterData.Projectile);

            List<Node> children = new()
            {
                new ActionSkillUseable(data.interval, data.attackableDistance),
                new ActionStop(),
                new ActionPlayAudio(MonsterAudioType.ProjectileAttack, true, false),
                new ActionPlayAnimation(AnimationType.PROJECTILE_CAST,true),
                new ActionPlayAnimation(AnimationType.PROJECTILE_FIRE, false),
                new ActionHold(data.animationHold),
                new ActionPlayAudio(MonsterAudioType.ProjectileFire),
                new ActionExecuteAttack(MonsterSkill.Projectile),
                new ActionSetAgent(tree.GetBTData<Vector3>(BTData.v3PlayerPosition)),
            };

            SetChildren(children);
        }
    }
}
```

4-4. ActionNode

4-4.1 공통

- ActionAssertBoolean

```
namespace Scripts.BehaviourTrees.Monster
{
    public class ActionAssertBoolean : MonsterNode
    {
        private bool boolean;
        public ActionAssertBoolean(bool boolean)
        {
            this.boolean = boolean;
        }
        public override NodeState Evaluate()
        {
            if (boolean) return NodeState.SUCCESS;
            else return NodeState.FAILURE;
        }
    }
}
```

- ActionDetect

```
namespace Scripts.BehaviourTrees.Monster
{
    public class ActionDetect : MonsterNode
    {
        private DetectType type;

        public ActionDetect(DetectType type)
        {
            this.type = type;
        }
        public override NodeState Evaluate()
        {
            switch (type)
            {
                case DetectType.PLAYER:
                    if
(tree.GetComponentData<DetectAI>(MonsterComponentData.PlayerDetectAI).IsDetected)
                        return NodeState.SUCCESS;
                    break;
                case DetectType.CHASE:
                    if
(tree.GetComponentData<DetectAI>(MonsterComponentData.ChaseDetectAI).IsDetected)
                        return NodeState.SUCCESS;
                    break;
            }

            return NodeState.FAILURE;
        }
    }
}
```

- ActionHold

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionHold : MonsterNode
    {
        private float holdTime;
        private float accumTime=0;
        public ActionHold(float holdTime= 1.0f) : base()
        {
            this.holdTime = holdTime;
```

```
        }
        protected override void OnStart()
        {
            accumTime = 0;
        }
        public override NodeState Evaluate()
        {
            if (accumTime >= holdTime) return NodeState.SUCCESS;
            accumTime += Time.deltaTime;
            return NodeState.RUNNING;
        }
    }
}
```

- ActionPlayAnimation

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionPlayAnimation : MonsterNode
    {
        Animator animator;
        private AnimationType animation;
        private bool waitToEnd;
        private float animationLength;
        private float accumTime;

        public ActionPlayAnimation(AnimationType animation, bool waitToEnd = false)
        {
            if(animator==null)
                animator = tree.GetComponentData<Animator>(MonsterComponentData.ANIMATOR);
            if (animator == null)
                DebugNull(transform, MonsterComponentData.ANIMATOR);

            this.animation = animation;
            this.waitToEnd = waitToEnd;
        }
        protected override void OnStart()
        {
            animator.SetTrigger(animation.ToString());

            if (waitToEnd)
            {
                accumTime = 0;
                AnimatorClipInfo[] clipInfo = animator.GetCurrentAnimatorClipInfo(0);
                animationLength = clipInfo[0].clip.length;
            }
        }
        public override NodeState Evaluate()
        {
            if(waitToEnd)
            {
                if (accumTime <= animationLength)
                {
                    accumTime += Time.deltaTime;
                    return NodeState.RUNNING;
                }
                else return NodeState.SUCCESS;
            }

            return NodeState.SUCCESS;
        }
    }
}
```

- ActionPlayAudio

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionPlayAudio : MonsterNode
    {
        private AudioSource audioSource;
        private MonsterAudioController audioController;
        private MonsterAudioType audioType;
        private bool isLoop;
        private bool isInteruptable;

        public ActionPlayAudio(MonsterAudioType audioType, bool isInteruptable = true, bool isLoop =
true) : base()
        {
            if (audioSource == null)
                audioSource = tree.GetComponentData<AudioSource>(MonsterComponentData.AUDIO);
            if (audioSource == null)
                DebugNull(transform, MonsterComponentData.AUDIO);

            if (audioController == null)
                audioController =
tree.GetComponentData<MonsterAudioController>(MonsterComponentData.AUDIO_CON);
            if (audioController == null)
                DebugNull(transform, MonsterComponentData.AUDIO_CON);

            this.audioType = audioType;
            this.isLoop = isLoop;
            this.isInteruptable = isInteruptable;
        }

        public override NodeState Evaluate()
        {
            if (isLoop)
                audioSource.loop = true;
            else audioSource.loop = false;

            if (!isInteruptable)
            {
                if (audioSource.isPlaying)
                    return NodeState.SUCCESS;
            }

            AudioClip clip = audioController.GetAudio(audioType);
            if(clip==null)
            {
                Debug.Log(transform.name + "Try To Play Audio Does Not Have: " +
audioType.ToString());
                return NodeState.FAILURE;
            }
            audioSource.clip = clip;
            audioSource.Play();

            return NodeState.SUCCESS;
        }
    }
}
```

- ActionPlayParticle

```
namespace Scripts.BehaviourTrees.Monster
{
    public class ActionPlayParticle : MonsterNode
    {
        private MonsterParticleController particleController;
        private MonsterParticleType particleType;

        public ActionPlayParticle(MonsterParticleType particleType)
        {
```

```
            particleController =
tree.GetComponentData<MonsterParticleController>(MonsterComponentData.PARTICLE);
            this.particleType = particleType;
        }
        public override NodeState Evaluate()
        {
            particleController.PlayParticle(particleType);

            return NodeState.SUCCESS;
        }
    }
}
```

- ActionSetAgent

```
using UnityEngine;
using UnityEngine.AI;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionSetAgent : MonsterNode
    {
        private NavMeshAgent agent;
        private Vector3 destination;

        public ActionSetAgent(Vector3 destination)
        {
            agent = tree.GetComponentData<NavMeshAgent>(MonsterComponentData.AGENT);
            if (agent == null)
                DebugNull(transform, MonsterComponentData.AGENT);

            this.destination = destination;
        }
        public override NodeState Evaluate()
        {
            agent.destination = destination;
            return NodeState.SUCCESS;
        }
    }
}
```

- ActionSetBoolean

```
namespace Scripts.BehaviourTrees.Monster
{
    public class ActionSetBoolean : MonsterNode
    {
        private BTData data;
        private bool boolean;

        public ActionSetBoolean(BTData data, bool boolean)
        {
            this.data = data;
            this.boolean = boolean;
        }
        public override NodeState Evaluate()
        {
            tree.SetBTData<bool>(data, boolean);
            return NodeState.SUCCESS;
        }
    }
}
```

- ActionStop

```
using UnityEngine.AI;

namespace Scripts.BehaviourTrees.Monster
```

```
{
    public class ActionStop : MonsterNode
    {
        private NavMeshAgent agent;

        public ActionStop()
        {
            agent = tree.GetComponentData<NavMeshAgent>(MonsterComponentData.AGENT);
        }
        public override NodeState Evaluate()
        {
            agent.destination = transform.position;
            return NodeState.SUCCESS;
        }
    }
}
```

- ActionUpdateData

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionUpdateData : MonsterNode
    {
        public ActionUpdateData() { }

        public override NodeState Evaluate()
        {
            Vector3 playerPosition =
tree.GetMonsterData<GameObject>(MonsterData.v3SpawnPosition).transform.position;
            tree.SetBTData<Vector3>(BTData.v3PlayerPosition, playerPosition);

            float playerDistance = Vector3.SqrMagnitude(playerPosition - transform.position);
            tree.SetBTData<float>(BTData.fPlayerDistanceSqr, playerDistance);

            float spawnDistance =
Vector3.SqrMagnitude(tree.GetMonsterData<Vector3>(MonsterData.v3SpawnPosition));
            tree.SetBTData<float>(BTData.fSpawnDistanceSqr, spawnDistance);

            if (spawnDistance >
tree.GetMonsterData<MonsterStatData>(MonsterData.MonsterStat).overtravelDist)
                tree.SetBTData<bool>(BTData.bOvertraveld, true);

            return NodeState.SUCCESS;
        }
    }
}
```

4-4.2 플레이어 미감지

- ActionReturnSpawnPosition

```
namespace Scripts.BehaviourTrees.Monster
{
    public class ActionReturnSpawnPosition : MonsterNode
    {
        public override NodeState Evaluate()
        {
            float distance = tree.GetBTData<float>(BTData.fSpawnDistanceSqr);

            if (distance > 0.5f)
                return NodeState.SUCCESS;

            else return NodeState.RUNNING;
        }
    }
}
```

4-4.3 움직임

- ActionCheckPlayerIsNear

```
namespace Scripts.BehaviourTrees.Monster
{
    public class ActionCheckPlayerIsNear : MonsterNode
    {
        public override NodeState Evaluate()
        {
            float playerDistance = tree.GetBTData<float>(BTData.fPlayerDistanceSqr);
            float playerStopDistance =
tree.GetMonsterData<MonsterStatData>(MonsterData.MonsterStat).stopDistance;

            if (playerDistance < playerStopDistance)
                return NodeState.SUCCESS;

            return NodeState.FAILURE;
        }
    }
}
```

- ActionLookPlayer

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionLookPlayer : MonsterNode
    {
        private float accumTime;
        private const float angle = 90.0f;
        private const float radius = 5.0f;

        public override NodeState Evaluate()
        {
            Vector3 playerPos = tree.GetBTData<Vector3>(BTData.v3PlayerPosition);
            Vector3 monsterPos = transform.position;
            Vector3 interV = playerPos - monsterPos;

            float dot = Vector3.Dot(interV.normalized, transform.forward.normalized);
            float theta = Mathf.Acos(dot);
            float degree = Mathf.Rad2Deg * theta;

            if (degree <= angle / 2.0f)
            {
                interV.y = 0;
                if (interV.sqrMagnitude <= radius * radius)
                    return NodeState.SUCCESS;
            }
            else
            {
                Quaternion targetRotation;
                Vector3 directionVector = interV;
                directionVector.y = 0;
                directionVector.Normalize();
                targetRotation = Quaternion.LookRotation(directionVector, Vector3.up);

                transform.rotation= Quaternion.Slerp(transform.rotation, targetRotation,
Time.deltaTime * 90.0f);
                accumTime += Time.deltaTime;

                if (accumTime >= 0.5f) return NodeState.FAILURE;
            }

            return NodeState.RUNNING;
```

```
        }
    }
}
```

- ActionPatrolToPoints

```
using UnityEngine;
using UnityEngine.AI;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionPatrolToPoint : MonsterNode
    {
        NavMeshAgent agent;
        Vector3[] patrolPoints;
        private int patrolNum;
        private int patrolQuant;

        public ActionPatrolToPoint()
        {
            if (patrolPoints == null)
                patrolPoints =
tree.GetComponentData<Vector3[]>(MonsterComponentData.PATROL_POINTS);
            if (patrolPoints == null)
                DebugNull(transform, MonsterComponentData.PATROL_POINTS);

            if (agent == null)
                agent =
tree.GetComponentData<NavMeshAgent>(MonsterComponentData.AGENT);
            if (agent == null)
                DebugNull(transform, MonsterComponentData.AGENT);

            patrolQuant = patrolPoints.Length;
            patrolNum = 0;
        }
        public override NodeState Evaluate()
        {
            agent.destination = patrolPoints[patrolNum];
            float distance = Vector3.SqrMagnitude(transform.position -
patrolPoints[patrolNum]);

            if(distance<0.1f)
            {
                patrolNum++;
                if (patrolNum == patrolQuant)
                    patrolNum = 0;

                return NodeState.SUCCESS;
            }

            return NodeState.RUNNING;
        }

    }
```

```
}
```

4-4.4 공격

- ActionExecuteAttack

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionExecuteAttack : MonsterNode
    {
        MonsterSkill monsterSkill;
        MonsterBase monster;

        public ActionExecuteAttack(MonsterSkill monsterSkill)
        {
            if(monster==null)
            monster = tree.GetComponentData<MonsterBase>(MonsterComponentData.MONSTER);
            if (monster == null)
                DebugNull(transform, MonsterComponentData.MONSTER);

            this.monsterSkill = monsterSkill;
        }
        public override NodeState Evaluate()
        {
            if (monster.ExecuteAttack(monsterSkill))
                return NodeState.SUCCESS;

            Debug.LogFormat("{0} Is Trying To Attack With {1} That Does Not have : {0}, {1}",
transform.name, monsterSkill.ToString());
            return NodeState.FAILURE;
        }
    }
}
```

- ActionRunToPlayer

```
using UnityEngine;
using UnityEngine.AI;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionRunToPlayer : MonsterNode
    {
        private RunToPlayerData skill;
        private NavMeshAgent agent;
        private float originalSpeed;
        private float accumTime;

        public ActionRunToPlayer()
        {
            skill = tree.GetMonsterData<RunToPlayerData>(MonsterData.RunToPlayer);
            if (agent == null)
                agent = tree.GetComponentData<NavMeshAgent>(MonsterComponentData.AGENT);
            if (agent == null)
                DebugNull(transform, MonsterComponentData.AGENT);

            originalSpeed = tree.GetMonsterData<MonsterStatData>(MonsterData.MonsterStat).speed;
            accumTime = 0.0f;
        }
        protected override void OnStart()
        {
            agent.speed = skill.speed;
            isStarted = true;
        }
        public override NodeState Evaluate()
        {
```

```
            if (!isStarted)
                OnStart();

            Vector3 playerPos = tree.GetBTData<Vector3>(BTData.v3PlayerPosition);
            agent.destination = playerPos;
            float distance = Vector3.SqrMagnitude(playerPos - transform.position);

            if(accumTime<skill.duration)
            {
                if (distance < skill.stopDistance)
                {
                    OnExit();
                    return NodeState.SUCCESS;
                }
                accumTime += Time.deltaTime;
                return NodeState.RUNNING;
            }

            OnExit();
            return NodeState.SUCCESS;
        }

        protected override void OnExit()
        {
            agent.speed = originalSpeed;
            accumTime = 0.0f;
            isStarted = false;
        }
    }
}
```

- ActionSkillUseable

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionSkillUseable : MonsterNode
    {
        float attackableDistance = 0.0f;
        float interval = 0.0f;
        private float lastTime;

        public ActionSkillUseable(float interval, float attackableDistance)
        {
            this.interval = interval;
            this.attackableDistance = attackableDistance;

            lastTime = 0.0f;
        }
        public override NodeState Evaluate()
        {
            float playerDistanceSqr = tree.GetBTData<float>(BTData.fPlayerDistanceSqr);

            if (playerDistanceSqr > attackableDistance)
                return NodeState.FAILURE;
            if (Time.time - lastTime < interval)
                return NodeState.FAILURE;

            lastTime = Time.time;

            return NodeState.SUCCESS;
        }
    }
}
```

4-5. Composite Node

- Select

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class Select : Node
    {
        private int current;
        public Select() : base() { }
        public Select(List<Node> childrens) : base(childrens) { }

        protected override void OnStart()
        {
            current = 0;
            isStarted = true;
        }
        public override NodeState Evaluate()
        {
            if (!isStarted)
                OnStart();
            for(int i=current;i<children.Count;i++)
            {
                current = i;
                var child = children[current];

                switch (child.Evaluate())
                {
                    case NodeState.FAILURE:
                        continue;
                    case NodeState.SUCCESS:
                        return NodeState.SUCCESS;
                    case NodeState.RUNNING:
                        return NodeState.RUNNING;
                    default:
                        continue;
                }
            }

            return NodeState.FAILURE;
        }
    }
}
```

- Sequence

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class Sequence : MonsterNode
    {
        public Sequence() : base() { }
        public Sequence(List<Node> childrens) : base(childrens) { }
        public override NodeState Evaluate()
        {
            bool anyChildIsRunning = false;

            foreach(Node child in children)
            {
                switch(child.Evaluate())
                {
                    case NodeState.FAILURE:
                        return NodeState.FAILURE;
                    case NodeState.SUCCESS:
                        return NodeState.SUCCESS;
```

```
                    case NodeState.RUNNING:
                        anyChildIsRunning = true;
                        continue;
                    default:
                        return NodeState.SUCCESS;
                }
            }

            return anyChildIsRunning ? NodeState.RUNNING : NodeState.SUCCESS;
        }
    }
}
```

4-6. Decorator

- Inverter

```
namespace Scripts.BehaviourTrees.Monster
{
    public class Inverter : Node
    {
        public Inverter(Node node) : base(node){}
        public override NodeState Evaluate()
        {
            switch (children[0].Evaluate())
            {
                case NodeState.RUNNING:
                    return NodeState.RUNNING;
                case NodeState.SUCCESS:
                    return NodeState.FAILURE;
                case NodeState.FAILURE:
                    return NodeState.SUCCESS;
            }

            return NodeState.FAILURE;
        }

    }
}
```

- Repeater

```
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class Repeater : Node
    {
        bool repeatOnSuccess;
        bool repeatOnFailure;
        public Repeater(Node child, bool repeatOnSuccess = true, bool repeatOnFailure = true) :
base(child)
        {
            this.repeatOnFailure = repeatOnFailure;
            this.repeatOnSuccess = repeatOnSuccess;
        }
        public Repeater(List<Node> childrens, bool repeatOnSuccess = true, bool repeatOnFailure =
true) : base(childrens)
        {
            this.repeatOnFailure = repeatOnFailure;
            this.repeatOnSuccess = repeatOnSuccess;
        }
        public override NodeState Evaluate()
        {
            if (children.Count != 1)
            {
                Debug.Log(transform.name + "Repeater Node Has No or More Than 1 Child");
                return NodeState.FAILURE;
```

```
            }
            switch (children[0].Evaluate())
            {
                case NodeState.RUNNING:
                    return NodeState.RUNNING;
                case NodeState.SUCCESS:
                    if (repeatOnSuccess) return NodeState.RUNNING;
                    return NodeState.SUCCESS;
                case NodeState.FAILURE:
                    if (repeatOnFailure) return NodeState.FAILURE;
                    return NodeState.FAILURE;
            }

            return NodeState.RUNNING;
        }

    }
}
```

5. 기타

5-1. MonsterData Class

```
using System.Collections.Generic;
using Channels.Combat;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterStatData
    {
        public MonsterType monsterType;
        public string name;
        public float HP;
        public float speed;
        public float rotationSpeed;
        public float detectPlayerDist;
        public float detectChaseDist;
        public float overtravelDist;
        public float overtravelReturnSpeed;
        public float stopDistance;
        public float weeknessRatio;
        public float respawnTime;
        public List<int> itemTableNum;
        public Vector3 spawnPosition;
    }
    public class RunToPlayerData
    {
        public float interval;
        public float attackableDistance;
        public float speed;
        public float duration;
        public float stopDistance;
    }
    public class MeleeAttackData
    {
        public CombatType combatType;

        public int damage;
        public float interval;
        public float attackableDistance;
        public float duration;
        public float animationHold;

        public float colliderDuration;
        public Vector3 colliderOffset;
        public Vector3 colliderSize;
    }
```

```
    public class WeaponAttackData
    {
        public CombatType combatType;

        public int damage;
        public float interval;
        public float attackableDistance;
        public float duration;
        public float animationHold;

        public float angle;
        public float radius;
    }
    public class FleeSkillData
    {
        public float fleeSpeed;
        public float interval;
        public float attackableDistance;
        public float duration;
        public float fleeDistance;
        public float animationHold;
    }
    public class ProjectileAttackData
    {
        public CombatType combatType;
        public string name;

        public float damage;
        public float interval;
        public float attackableDistance;
        public float duration;
        public float animationHold;
        public float projectileSpeed;
    }
}
```

5-2. BehaviourTree Enum

```
namespace Scripts.BehaviourTrees.Monster
{
    public enum MonsterComponentData
    {
        MONSTER,
        AGENT,
        AUDIO,
        AUDIO_CON,
        TRANSFORM,
        ANIMATOR,
        PATROL_POINTS,
        PARTICLE,
        MONSTER_CENTER,

        PlayerDetectAI,
        ChaseDetectAI,
    }
    public enum BTData
    {
        bOnSpawnPosition,
        bOvertraveld,
        bReturning,

        fPlayerDistanceSqr,
        fSpawnDistanceSqr,

        v3PlayerPosition,

        iCurrentHP,
    }
    public enum MonsterData
```

```
    {
        v3SpawnPosition,

        // Datas
        MonsterStat,
        RunToPlayer,
        Melee,
        Weapon,
        Flee,
        Projectile,
    }
    public enum MonsterSkill
    {
        RunToPlayer,
        Melee,
        Projectile,
        Weapon,
        Flee,
    }
    public enum DetectType
    {
        PLAYER,
        CHASE,
    }
    public enum AnimationType
    {
        IDLE,
        STANDUP,
        WALK,
        SIT,
        IDLE_ATTACK,
        RUN,
        MELEE,
        WEAPON,
        FLEE,
        PROJECTILE_CAST,
        PROJECTILE_FIRE,
    }
    public enum MonsterType
    {
        NormalSkeleton,
        AdventureSekeleton,
        WizardSkeleton,
        GuildguardSkeleton
    }
}
```